Classification and Analysis of Computer Network Traffic

Bujlow, Tomasz

Publication date:
2014

Document Version
Early version, also known as pre-print

Classification and Analysis of Computer Network Traffic

TOMASZ BUJLOW

Networking & Security
Department of Electronic Systems
Aalborg University

Classification and Analysis of Computer Network Traffic

Tomasz Bujlow

Networking & Security
Department of Electronic Systems
Aalborg University

Tomasz Bujlow. *Classification and Analysis of Computer Network Traffic.*

PHD THESIS

Date of defense: June 3, 2014

Distribution:
Aalborg University
Department of Electronic Systems
Networking & Security
Fredrik Bajers Vej 7 A4
DK-9220 Aalborg
Denmark
Phone: +45 9940 8616
Fax: +45 9940 9840
netsec@es.aau.dk

ISBN: 978-87-71520-30-9

DEN EUROPÆISKE UNION
Den Europæiske Fond for Regionaludvikling
Vi investerer i din fremtid

Abstract

Traffic monitoring and analysis can be done for multiple different reasons: to investigate the usage of network resources, assess the performance of network applications, adjust Quality of Service (QoS) policies in the network, log the traffic to comply with the law, or create realistic models of traffic for academic purposes. We define the objective of this thesis as finding a way to evaluate the performance of various applications in a high-speed Internet infrastructure. To satisfy the objective, we needed to answer a number of research questions. The biggest extent of them concern techniques for traffic classification, which can be used for nearly real-time processing of big amounts of data using affordable CPU and memory resources. Other questions are related to methods for real-time estimation of the application Quality of Service (QoS) level based on the results obtained by the traffic classifier. This thesis is focused on topics connected with traffic classification and analysis, while the work on methods for QoS assessment is limited to defining the connections with the traffic classification and proposing a general algorithm.

We introduced the already known methods for traffic classification (as by using transport layer port numbers, Deep Packet Inspection (DPI), statistical classification) and assessed their usefulness in particular areas. We found that the classification techniques based on port numbers are not accurate anymore as most applications use dynamic port numbers, while DPI is relatively slow, requires a lot of processing power, and causes a lot of privacy concerns. Statistical classifiers based on Machine Learning Algorithms (MLAs) were shown to be fast and accurate. At the same time, they do not consume a lot of resources and do not cause privacy concerns. However, they require good quality training data. We performed substantial testing of widely used DPI classifiers (PACE, OpenDPI, L7-filter, nDPI, Libprotoident, and NBAR) and assessed their usefulness in generating ground-truth, which can be used as training data for MLAs. Our evaluation showed that the most accurate classifiers (PACE, nDPI, and Libprotoident) do not provide any consistent output – the results are given on a mix of various levels: application, content, content container, service provider, or transport layer protocol. On the other hand, L7-filter and NBAR provide results consistently on the application level, however, their accuracy is too low to consider them as tools for generating the ground-truth. We also contributed to the open-source community by improving the accuracy of nDPI and designing the future enhancements to make the classification consistent.

Because the existing methods were shown to not be capable of generating the proper training data, we built our own host-based system for collecting and labeling of network data, which depends on volunteers and, therefore, was called by us Volunteer-Based

System (VBS). The client registers the information about all the packets transferred through any network interface of the machine on which it is installed. The packets are grouped into flows, which are labeled by the process name obtained from the system sockets. The detailed statistics about the network flows give an overview how the network is utilized. The data collected by VBS can be used to create realistic traffic profiles of the selected applications, which can server as the training data for MLAs.

We assessed the usefulness of C5.0 Machine Learning Algorithm (MLA) in the classification of computer network traffic. We showed that the application-layer payload is not needed to train the C5.0 classifier to be able to distinguish different applications in an accurate way. Statistics based on the information accessible in the headers and the packet sizes are fully sufficient to obtain high accuracy. We also contributed by defining the sets of classification attributes for C5.0 and by testing various classification modes (decision trees, rulesets, boosting, softening thresholds) regarding the classification accuracy and the time required to create the classifier.

We showed how to use our VBS tool to obtain per-flow, per-application, and per-content statistics of traffic in computer networks. Furthermore, we created two datasets composed of various applications, which can be used to assess the accuracy of different traffic classification tools. The datasets contain full packet payloads and they are available to the research community as a set of PCAP files and their per-flow description in the corresponding text files. The included flows were labeled by VBS.

We also designed and implemented our own system for multilevel traffic classification, which provides consistent results on all of the 6 levels: *Ethernet*, *IP protocol*, *application*, *behavior*, *content*, and *service provider*. The *Ethernet* and *IP protocol* levels are identified directly based on the corresponding fields from the headers. The *application* and *behavior* levels are assessed by a statistical classifier based on C5.0 Machine Learning Algorithm. Finally, the *content* and *service provider* levels are identified based on IP addresses. The system is able to deal with unknown traffic, leaving it unclassified on all the levels, instead of assigning the traffic to the most fitting class. Our system was implemented in Java and released as an open-source project.

Finally, we created a method for assessing the Quality of Service in computer networks. The method relies on VBS clients installed on a representative group of users from the particular network. The per-application traffic profiles obtained from the machines belonging to the volunteers are used to train the C5.0 Machine Learning based tool to recognize the selected applications in any point of the network. After the application is being identified, the quality of the application session can be assessed. For that purpose, we proposed a hybrid method based on both passive and active approaches. The passive approach can be used to assess jitter, burstiness, download and upload speeds, while

the active one is needed when we want to measure delay or packet loss.

Resumé

Der er adskillige grunde til at foretage trafikovervågning og trafikanalyse: For at undersøge hvordan netværksressourcerne anvendes, vurdere hvor godt forskellige applikationer afvikles, justere Quality of Service (QoS) politikker i netværket, logge trafik for at overholde lovgivning, eller indsamle data med henblik på at skabe realistiske modeller for netværkstrafik. Vi definerer at målet med denne afhandling er at finde en måde at vurdere hvor godt forskellige applikationer afvikles, i en højhastigheds Internet infrastruktur. For at nå dette mål er det nødvendigt at besvare en række forskningsspørgsmål, der primært omhandler teknikker til trafikklassificering som bruges til næsten-realtidsbehandling af store mængder af data, uden at stille alt for store krav til beregningsressourcer som CPU og hukommelse. Andre spørgsmål relaterer sig til metoder til realtidsestimering af applikationernes Quality of Service (QoS) niveau baseret på resultater opnået gennem trafikklassifikation. Denne afhandling fokuserer især på emner i forbindelse med trafikklassifikation og analyse, hvorimod arbejdet med metoder til QoS vurdering er afgrænset til at definere forbindelserne ved hjælp af trafikklassifikation, og foreslå en generel algoritme.

Vi introducerede allerede kendte metoder til trafikklassifikation (portnumre på transportlaget, Deep Packet Inspection (DPI) og statistisk klassifikation), og vurderede deres brugbarhed på forskellige områder. Vi nåede frem til at klassfikationsteknikker baseret på portnumre ikke længere er præcise, da mange mange applikationer bruger dynamiske portnumre, mens DPI er relativt langsomt, kræver mange CPU ressourcer, og også giver anledning til betænkelighed i forhold til sikring af privatliv og følsome data. Vi viste at statistiske klassifikatorer baseret på Maskin Lærings Algoritmer (MLAs) er både hurtige og præcise. Samtidig kræver de kun begrænsede ressourcer, og giver ikke anledning til betænkeligheder i forhold til datahåndtering/privatliv. De kræver derimode gode træningsdata. Vi gennemført grundige test af udbredte DPI klassifikatorer (PACE, OpenDPI, L7-filter, nDPI, Libprotoident og NBAR) og vurderet deres brugbarhed i forhold til at generere ground-truth, som kan bruges til klassfikation af data for MLAs. Vores undersøgelser viste at de mest præcise klassifikatorer (PACE, nDPI og Libprotoident) ikke giver konsistente output – resultaterne præsenteres i en blanding af forskellige niveauer: Applikation, indhold, indholdscontainer, serviceudbyder, or transportlagsprotokol. På den anden side giver L7-filter og NBAR konsekvent resultater på applikationsniveau. Disse er imidlertid for upræcise til at kunne bruges til at generere ground-truth. Vi bidrog også til det open-source samfund ved at forbedre nøjagtigheden af nDPI og designe fremtidige forbedringer for at gøre klassificeringen konsekvent.

Da de eksisterende metoder ikke er i stand til at generere korrekte træningsdata byggede vi vores eget host-baserede system til indsamling og mærkning af netværksdata. Systemet afhænger af frivillig, og kaldes derfor Volunteer-Based System (VBS). Klienter registrerer information om alle pakker der sendes gennem et hvilket som helst netværksinterface på maskine med VBS installeret. Disse pakker grupperes så i flows, og mærkes med procesnavnet der fås fra system sockets. De detaljeres netværksflow-statistikker giver et overblik over hvordan netværket anvendes. Desuden kan data indsamlet via VBS bruges til at danne realistiske trafikprofiler for udvalgte applikationer, hvilket kan anvendes som træningsdata for MLAs.

Vi vurderede brugbarheden af C5.0 Maskin Lærings Algoritmen (MLA) i forhold til klassifikation af computernetværkstrafik. Vi viste at applikations-lag payload ikke er nødvendigt for at træne C5.0 klassifikatoren for at kunne skelne mellem forskellige applikationer på en præcis måde. Statistik baseret på den information der er tilgængeligt i headers og ud fra pakkestørrelser er helt tilstrækkeligt til at opnå en høj præcision. Vi har også bidraget ved at definere klassifikationsattributter for C5.0, og ved at teste forskellige klassifikationstilstande (decision trees, rulesets, boosting, softening thresholds) i forhold til klassifikationspræcision og hvor lang tid det tager at generere klassifikatoren.

Vi viste hvordan vores VBS værktøj kan bruges til at indsamle per-flow, per-applikation og per-indhold statistik for trafik i computernetværk. Derudover har vi tilvejebragt to datasæt bestående af forskellige applikationer, hvilket kan bruges til at vurdere præcisionen af forskellige trafikklassifikationsværktøjer. Disse datasæt indeholder komplette pakker med payload, og er tilgængelige for forskere som PCAP filer, og deres per-flow beskrivelser i tilhørende tekstfiler. De inkluderede flows er mærket med VBS.

Vi har også designet og implementeret vores eget system til multiniveau trafik klassifikation, der giver konsistente resultater på alle 6 niveauer: *Ethernet*, *IP protocol*, *application*, *behavior*, *content* og *service provider*. Niveauerne *Ethernet* og *IP protocol* er identificeret direkte baseret på de tilsvarende felter i headerne. Niveauerne *application* og *behavior* er fastlagt af en statistisk klassifikator baseret på C5.0 Maskin Lærings Algoritmen. Nivauerne *content* og *service provider* er identificeret på baggrund af IP-adresser. Systemet er i stand til at håndtere ukendt trafik, og lade det forblive uklassificeret på alle niveauer i stedet for at vælge den nærmeste klasse. Vores system er implementeret i Java og frigivet som open-source projekt.

Vi har også udviklet en metode til at vurdere Quality of Service i computernetværk. Denne metode bygger på VBS-klienter installeret ved en repræsentativ gruppe af brugere på et specifikt netværk. De per-application trafikprofiler der fås fra de frivllige bruges til at træne et C5.0 Maskin Lærings baseret værktøj, så det kan genkende de valgte

applikationer et hvilket som helst sted i netværket. Efter applikationen er blevet identificeret kan kvaliteten af en applikations-session vurderes. Til det formål foreslår vi en hybrid metode baseret på både aktive og passive tilgange. Den passive tilgang kan bruges til at vurdere jitter, burstiness, download og upload hastigheder, mens den aktive er nødvendig når vi vil måle forsinkelsestid og pakketab.

Acknowledgments

The work presented in this thesis was made possible by many people. First of all, I would like to thank my supervisor Jens Myrup Pedersen and my co-supervisors Tahir Riaz and Pere Barlet-Ros for being excellent mentors and for their guidance, feedback, and patience throughout my PhD studies. Secondly, special thanks to my PhD Assessment Committee members, Christian Kreibich, Josep Solé-Pareta, and Reza Tadayoni, for valuable discussions and their contributions in improving this thesis.

I would like to thank my colleagues from the Department of Electronic Systems for being a truly wonderful bunch of people. Many thanks to the co-authors of the publications in this thesis and all the other people from the Networking and Security research group, who by other means contributed to my work, especially to Kartheepan Balachandran, Sara Ligaard Nørgaard Hald, Michael Jensen, and Matija Stevanovic. A special thanks to Dorthe Sparre, the section administrator and secretary, who was always offering her help.

I am very grateful to Josep Solé-Pareta, a professor from Universitat Politècnica de Catalunya (UPC) in Barcelona for hosting me 3-months at UPC in an excellent research environment. The collaboration with Valentín Carela-Español significantly contributed to the knowledge used in creating this thesis.

Thank you to Luca Deri and Alfredo Cardigliano from ntop[1] and to Marco Mellia from Politecnico di Torino for the time and discussions, which led to broaden my knowledge about Deep Packet Inspection and thus directly contributed to the content of the thesis. Thanks to Laura Leone, a skilled life coach[2], for all the support during my stay in Italy.

I also wish to thank Indira Paudel and Badii Jouaber from the Department of Networks and Mobile Multimedia Services[3] at Télécom SudParis for the collaboration and valuable discussions about the Quality of Service in wireless networks.

This work was funded by several partners. The PhD project was co-financed by the European Regional Development Fund (ERDF)[4] and Bredbånd Nord A/S[5], a regional

[1] See http://www.ntop.org/home/whos-behind-ntop/
[2] See http://www.lauraleone.it/
[3] See http://rs2m.telecom-sudparis.eu/en/
[4] See http://ec.europa.eu/regional_policy/thefunds/regional/index_en.cfm
[5] See http://www.bredbaandnord.dk/

fiber networks provider. The research performed during my stay in Barcelona was partly funded by the Spanish Ministry of Science and Innovation under contract TEC2011-27474 (NOMADS project) and by the Comissionat per a Universitats i Recerca del DIUE de la Generalitat de Catalunya (ref. 2009SGR-1140). I am also very grateful, for the possibility to install and test the software developed during the project, to Gimnazjum nr 3 z Oddziałami Integracyjnymi i Dwujęzycznymi imienia Karola Wojtyły w Mysłowicach[6], a high school in Poland, and to Bredbånd Nord A/S.

Last, but not least, I would like to thank my family and friends for their support. Special thank you to Gergana Todorova Todorova for all the extensive support and discussions about the life in general during the 3.5 years spent in Aalborg.

Aalborg, June 2014
Tomasz Bujlow

[6] See http://www.nr3.edu.pl/

Thesis Details

Thesis Title:	Classification and Analysis of Computer Network Traffic
Date of Defense:	June 3, 2014
PhD Student:	Tomasz Bujlow
Main Supervisor:	Jens Myrup Pedersen, Associate Professor, Aalborg University, Denmark
Co-Supervisors:	1. Tahir Riaz, Associate Professor, Aalborg University, Denmark 2. Pere Barlet-Ros, Associate Professor, Universitat Politècnica de Catalunya, Spain
PhD Assessment Committee:	1. Christian Kreibich, Senior Research Scientist, Networking and Security Group, International Computer Science Institute, Berkeley, California, USA 2. Josep Solé-Pareta, Full Professor, Universitat Politècnica de Catalunya, Spain 3. Reza Tadayoni, Associate Professor, Aalborg University, Denmark (Chairman)

List of Appended Papers

This thesis is based on the work presented in the following ten papers. Some of them were subjected to minor editorial changes. References to the papers are made using the roman numbers associated with the papers.

I Tomasz Bujlow, Kartheepan Balachandran, Sara Ligaard Nørgaard Hald, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for Research on the Internet Traffic. In the *TELFOR Journal*, volume 4, number 1, pages 2–7. September 2012. Accessible: http://journal.telfor.rs/Published/Vol4No1/Vol4No1.aspx.

II Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A Method for Classification of Network Traffic Based on C5.0 Machine Learning Algorithm. In the *Pro-*

ceedings of ICNC'12: 2012 International Conference on Computing, Networking and Communications (ICNC): Workshop on Computing, Networking and Communications, pages 244–248. IEEE, Maui, Hawaii, USA, February 2012. DOI: 10.1109/ICCNC.2012.6167418.

III Tomasz Bujlow, Sara Ligaard Nørgaard Hald, Tahir Riaz, and Jens Myrup Pedersen. A Method for Evaluation of Quality of Service in Computer Networks. In the *ICACT Transactions on the Advanced Communications Technology (TACT)*, volume 1, number 1, pages 17–25. Global IT Research Institute (GiRI), July 2012. Accessible: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6488383.

IV Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. Classification of HTTP Traffic Based on C5.0 Machine Learning Algorithm. In the *Proceedings of the Fourth IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Web-based Service Architectures (PEDISWESA 2012)*, pages 882–887. IEEE, Cappadocia, Turkey, July 2012. DOI: 10.1109/ISCC.2012.6249413.

V Jens Myrup Pedersen and Tomasz Bujlow. Obtaining Internet Flow Statistics by Volunteer-Based System. In the *Fourth International Conference on Image Processing & Communications (IP&C 2012), Image Processing & Communications Challenges 4, AISC 184*, pages 261–268. Springer Berlin Heidelberg, Bydgoszcz, Poland, September 2012. DOI: 10.1007/978-3-642-32384-3_32.

VI Tomasz Bujlow and Jens Myrup Pedersen. Obtaining Application-Based and Content-Based Internet Traffic Statistics. In the *Proceedings of the 6th International Conference on Signal Processing and Communication Systems (ICSPCS'12)*, pages 1–10. IEEE, Gold Coast, Queensland, Australia, December 2012. DOI: 10.1109/ICSPCS.2012.6507984.

VII Valentín Carela-Español, Tomasz Bujlow, and Pere Barlet-Ros. Is our Ground-Truth for Traffic Classification Reliable?. In the *Proceedings of the 15th Passive and Active Measurement Conference (PAM 2014), Proceedings Series: Lecture Notes in Computer Science 8362*, pages 98–108. Springer International Publishing Switzerland, Los Angeles, USA, March 2014. DOI: 10.1007/978-3-319-04918-2_10.

VIII Tomasz Bujlow and Jens Myrup Pedersen. Multilevel Classification and Accounting of Traffic in Computer Networks. Submitted for review to *Computer Networks* – a journal from Elsevier.

IX Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Independent Comparison of Popular DPI Tools for Traffic Classification. Submitted for review to *Computer Networks* – a journal from Elsevier.

X Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. nDPI: Open-Source High-Speed Deep Packet Inspection. To appear in the *Proceedings of the 10th International Wireless Communications & Mobile Computing Conference 2014 (IWCMC2014)*. IEEE, Nicosia, Cyprus, August 2014.

Comments on My Participation

I am responsible for the most of the written material and for carrying out all the experiments with the exception of the cases described below. My supervisor and advisors contributed by participation in discussions about the scope of the papers, methods used in the papers, and by providing comments on the papers throughout the writing process.

- In Paper I, Kartheepan Balachandran performed the evaluation of the changes performed by me to the Volunteer-Based System, which was originally described by Kartheepan Balachandran et al.[7]. In addition, he contributed by writing.
- In Paper I, Sara Ligaard Nørgaard Hald was fully responsible for the threat assessment part.
- In Paper III, Sara Ligaard Nørgaard Hald contributed by discussion about the idea, editorial changes and commenting.
- The scope of Paper V was defined in collaboration with Jens Myrup Pedersen, who wrote most of the text of the paper. I was responsible for performing the experiments and obtaining the results.
- Paper VII and Paper IX are based on the work done by me and Valentín Carela-Español. We were both involved in discussions, writing of the papers, and in the experiments, while there are some experimental parts, which were done only by one of us. I was responsible for designing and creating the system for collecting entire packets (together with their payload), storing the packets in the database, dumping them to PCAP files, and analyzing the classification logs. I was also responsible for the entire process of testing NBAR. Valentín Carela-Español was completely responsible for creating a software for testing PACE, OpenDPI, nDPI, L7-filter, and Libprotoident.
- In Paper X, I was responsible for proposing and implementing some enhancements to nDPI, testing the new version, and obtaining the results in terms of the accuracy. Apart from that, all the content is written by Luca Deri, Maurizio Martinelli, and Alfredo Cardigliano.

[7] Kartheepan Balachandran, Jacob Honoré Broberg, Kasper Revsbech, and Jens Myrup Pedersen. Volunteer-Based Distributed Traffic Data Collection System. In the Proceedings of the 12th International Conference on Advanced Communication Technology (ICACT 2010), volume 2, pages 1147–1152. IEEE, February 2010.

Other Papers

Apart from the papers included in this thesis, I am the main author of the following two conference papers and three technical reports. The conference papers were not included in the thesis, as all the content was subsequently used in our journal papers. The technical reports were not included due to their excessive length.

- Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for Classification of Traffic in Computer Networks. In the *Proceedings of the 19th Telecommunications Forum TELFOR 2011*, pages 210–213. IEEE, Belgrade, Serbia, November 2011. DOI: 10.1109/TELFOR.2011.6143528.

- Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A Method for Assessing Quality of Service in Broadband Networks. In the *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT)*, pages 826–831. IEEE, Phoenix Park, PyeongChang, Korea, February 2012. Accessible: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6174795.

- Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification. Technical report, Department of Computer Architecture (DAC), Universitat Politècnica de Catalunya (UPC), pages 1–108, June 2013. Accessible: https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2013,en.html.

- Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Extended Independent Comparison of Popular Deep Packet Inspection (DPI) Tools for Traffic Classification. Technical report, Department of Computer Architecture (DAC), Universitat Politècnica de Catalunya (UPC), pages 1–440, January 2014. Accessible: https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2014,en.html.

- Tomasz Bujlow and Jens Myrup Pedersen. A Practical Method for Multilevel Classification and Accounting of Traffic in Computer Networks. Technical report, Section for Networking and Security, Department of Electronic Systems, Aalborg University, pages 1–56, February 2014. Accessible: http://vbn.aau.dk/da/persons/tomasz-bujlow(c70a43e3-4879-4fd6-89fd-d4679bd168cf)/publications.html.

Contents

Summary

1 Introduction

One of the most important challenges in today's world is how to measure the performance of computer network infrastructures, when different types of networks are merged together. In the last few years, the data-oriented networks evolved into converged structures, in which real-time traffic, like voice calls or video conferences, is more and more important. The structure is composed of traditional cable or more modern fiber links, mobile and wireless networks. There are numerous methods for the measurement of Quality of Service (QoS) in current use, which provide the measurements both on the user side and in the core of the network. Internet Service Providers are interested in centralized measurements and detecting problems with particular customers before the customers start complaining about the problems, and if possible, before the problems are even noticed by the customers.

Each network carries data for numerous different kinds of applications. QoS requirements are dependent on the service. The main service-specific parameters are bandwidth, delay, jitter, and packet loss. Therefore, in order to provide detailed information about the quality level for the given service in the core of the network, we need to know, what kind of data is flowing in the network at the present time. Processing all the packets flowing in a high-speed network and examining their payload to get the application name is a very hard task, involving large amounts of processing power and storage capacity. Furthermore, numerous privacy and confidentiality issues can arise. A solution for this problem can be the use of Machine Learning Algorithms (MLAs), which depend on previously generated decision rules based on some statistical information about the traffic. In our research, we used one of the newest MLAs – C5.0. MLAs need very precise training sets to learn how to accurately classify the data, so the first issue to be solved was to find a way to collect high-quality training data.

In order to collect the necessary statistics and generate training sets for C5.0, a new system was developed, in which the major role is performed by volunteers. Client applications installed on their computers collect the detailed information about each flow passing through the network interface, together with the application name taken

from the description of system sockets. The information about each packet belonging to the flow is also collected. Our volunteer-based system guarantees obtaining of precise and detailed datasets, which can be successfully used to generate statistics used as the input to train MLAs and to generate accurate decision rules.

In the thesis, we also show an alternative strategy of traffic identification. We introduce nDPI, a high-performance traffic classification tool based on Deep Packet Inspection (DPI), for which we proposed numerous improvements, making it the only DPI tool able to consistently label the data.

The summary part of this thesis is structured as follows. At first, we introduce our research objective and methodology in Section 2. Then, in Section 4, we present the questions, which we needed to answer in order to achieve the final objective. Main contributions are described in in Section 3, while Section 5 summarizes the contents of each paper included in this thesis. In Section 6, we synthesize the impact of our research on the scientific community and our collaboration partners and in Section 7, we describe the potential of the future work to be done in the field of traffic analysis and classification.

2 Research Objective and Methodology

The research objective of this thesis can be defined as finding a way to evaluate the performance of applications in a high-speed Internet infrastructure. To satisfy the objective, we needed to answer a number of research questions. The biggest extent of them concern techniques for traffic classification, which can be used for nearly real-time processing of big amounts of data using affordable CPU and memory resources. Other questions are about methods for real-time estimation of the application Quality of Service (QoS) level based on the results obtained by the traffic classifier. This thesis is focused on topics connected with traffic classification and analysis, while the work on methods for QoS assessment is limited to defining the connections with the traffic classification and proposing a general algorithm.

There are 2 main methods to deal with research questions: analytical and experimental. The first one focuses on the mathematical approach to illustrate the questioned problem as a set of equations and formulas, while the second one uses simulation techniques or experiments based on the real data. However, it is very difficult to create a model, which includes all the existing parameters so it behaves like the real-life scenario. Therefore, the functionality of such models is quite limited. Simulations allow to test non-complex scenarios in a research environment, which is easier and faster than creating a mathematical model. However, the particular parts of the simulators are created with different precision, so in many cases not all of the actual properties are reflected by the simulator. This thesis in its entire whole relies on the last approach – experiments based on real data. That was made possible due to our collaboration with Bredbånd Nord, a Danish provider of fiber-based Internet connections. Studies performed on real

data are characterized by high diversity of the situations reflected by the data, high accuracy, and possibilities to analyze the information, which are not accessible in any other way. Such information includes IP addresses or HTTP headers. All the results presented in the attached papers were generated by performing experiments using real-life scenarios. Thanks to that, they are characterized by high significance for a practical appliance.

3 Main Contributions

This thesis contains several contributions, which are mostly distributed among the research papers. The main contributions are summarized below:

1. We designed, developed, and evaluated a host-based traffic monitoring tool, called Volunteer-Based System (VBS). This contribution is presented in Paper I. The system is an extension of the already existing *Volunteer-based Distributed Traffic Data Collection System* developed at Aalborg University during the previous years [1, 2]. It consists of clients installed on computers belonging to volunteers and of a server. The client registers the information about all the packets transferred through any network interface of the machine on which it is installed. The packets are grouped into flows based on 5-tuple: local and remote IP addresses, local and remote ports, and the transport layer protocol. Additionally, the process name obtained from the system sockets is registered and collected together with other information about the flow. Detailed information about each packet is collected as well: direction, size, state of the TCP flags, and timestamp. For the privacy reasons, the normal version of VBS does not collect the actual application-layer payload. However, another version of the system was developed and shown in Paper VII and Paper IX, where the full Ethernet frame is collected as well. VBS is a tool, which can be used for numerous useful tasks distributed among the research community and the industry:

    - The detailed statistics about the network flows give an overview how the network is utilized. That knowledge allows to better allocate the available bandwidth (by creating or improving the Quality of Service policies in the network), tune the network parameters on routers and switches, or to create special facilities for particular application or services (as proxy or cache servers, local DNS servers).
    - Users located in the same network can be compared and assigned to specific profiles, e.g. VoIP users, heavy downloaders, etc. The users can be directly targeted by recommending special Internet packages by their ISPs, as a low-latency one, but also low-throughput connection, or unlimited download rates

at night. Assigning these users to separate VLANs allows to control the traffic in a more efficient way, when one user does not interfere with another one.

- The data collected by VBS can be used to create realistic traffic profiles of the selected applications. They can be used to build application traffic generators, where the characteristics of the generated traffic match the real characteristics of the simulated applications.
- Collected HTTP flows are also marked by the type of the content, which they carry. This information can be used to discover the most often transferred type of traffic (as audio, video, other binary files, websites) from the particular hosts, and to improve the user's experience by tweaking the network settings.
- The traffic profiles can be used as the ground-truth for testing and building new network traffic classifiers. A special version of VBS, which collects full Ethernet frames, can be also used to evaluate the accuracy of Deep Packet Inspection classification tools.

2. We created a method for assessing the Quality of Service in computer networks. This contribution is presented in Paper III. The method relies on VBS clients installed on a representative group of users from the particular network. The per-application traffic profiles obtained from the machines belonging to the volunteers are used to train the C5.0 Machine Learning based tool to recognize the selected applications in any point of the network. The tool uses only attributes based on packet headers or some statistical parameters (as packet sizes), so there is no problem with users' privacy issues due to the inspection of the traffic. After the application is being identified, the quality of the application session can be assessed. For that purpose, we proposed a hybrid method based on both passive and active approaches. The passive approach can be used to assess jitter, burstiness, download and upload speeds, while the active one is needed when we want to measure delay or packet loss.

3. We assessed the usefulness of C5.0 Machine Learning Algorithm (MLA) in the classification of computer network traffic. This contribution is shown in Paper II and Paper IV. MLAs require good quality training data in order to be able to create accurate classification rules. Therefore, as the source of the training data, we used the information collected by VBS. The particular contributions associated with C5.0 classification are pointed out below:

   - In both papers, we showed that the application-layer payload is not needed to train the C5.0 classifier to be able to distinguish different applications in an accurate way. Statistics based on the information accessible in the headers and the packet sizes are fully sufficient to obtain high accuracy.

- In Paper II, the method was shown to reach 99.9 % accuracy while recognizing the traffic belonging to 7 different groups: Skype, FTP, BitTorrent, web browsing, web radio, interactive gaming, and SSH. In Paper IV, the method was shown to reach around 83 % accuracy while recognizing various types of content transported by HTTP.
- In both papers, we contributed by defining the sets of classification attributes for C5.0.
- In Paper II, we tested various classification modes (decision trees, rulesets, boosting, softening thresholds) regarding the classification accuracy and the time required to create the classifier.
- In Paper II, we also assessed the dependency between the number of training cases and the classification accuracy by C5.0.
- In Paper II, we showed how the number of packets in the sample from which the statistics are calculated influences the classification accuracy.
- In Paper IV, we showed how the information from the *content-type* field in HTTP headers can be used to split the transport-layer flows into parts transporting different files.
- In Paper IV, we compared statistical profiles of flows transmitting different types of HTTP contents. We showed that the profiles do not depend on the type of the transmitted content, but on the particular application or web service. We demonstrated that there is no difference between the profiles of transfers of video files (f.x. from YouTube) and transfer of other big binary files (as ZIPs) by HTTP and, therefore, different content types cannot be distinguished based on the statistical attributes. However, the paper shows that we can identify the content if it is associated with a particular protocol or behavior, as for example we can recognize video streaming (by RTMP protocol) or audio streaming (by both RTMP and HTTP).

4. We showed how to obtain per-flow, per-application, and per-content statistics of traffic in computer networks. This contribution is shown in Paper V and Paper VI. Specifically:

   - In Paper V, our VBS was used to obtain overall and per-user statistics based on flows. We showed the distribution of TCP and UDP flows, average lengths and durations of TCP and UDP flows, and the distribution of flows belonging to top 5 applications. Finally, the cumulative number of flows for each user over the time was shown on the graph to provide the overview of the users' network activity during several months.
   - In Paper VI, we extended our research by including statistics regarding the traffic volume. It enabled us to compare the number of flows and their

distribution with the corresponding amount of data for all monitored users altogether and for each user separately.

- In Paper VI, statistics regarding transferred files by HTTP were included in our evaluation. Therefore, we identified the most common types of files transferred by HTTP and presented which share of the overall HTTP traffic they represent.
- In Paper VI, we showed how by the monitoring of download and upload rates we can identify users, which have asymmetric Internet connections.

5. We created two datasets composed of different applications, which can be used to assess the accuracy of different traffic classification tools. The datasets contain full packet payloads and they are available to the research community as a set of PCAP files and their per-flow description in the corresponding text files. The included flows were labeled by VBS. Specifically:

   - The first dataset, shown in Paper VII, contains 1 262 022 flows, where 490 355 flows were labeled with the following applications and protocols: eDonkey, BitTorrent, FTP, DNS, NTP, RDP, NETBIOS, SSH, HTTP, and RTMP; the rest flows were left unlabeled.
   - The second dataset, shown in Paper IX, contains 767 690 flows labeled on a multidimensional level. The flows can be associated with one or more of 17 application protocols, 19 applications (also various configurations of the same application), and 34 web services. The included application protocols are: DNS, HTTP, ICMP, IMAP (STARTTLS and TLS), NETBIOS (name service and session service), SAMBA, NTP, POP3 (plain and TLS), RTMP, SMTP (plain and TLS), SOCKSv5, SSH, and Webdav. The included applications (and their configurations) are: 4Shared, America's Army, BitTorrent clients (plain and encrypted modes), Dropbox, eMule clients (plain and obfuscated modes), Freenet, FTP clients (active and passive modes), iTunes, League of Legends, Pando Media Booster, PPLive, PPStream, RDP, Skype (including audio conversations, file transfers, video conversations), Sopcast, Spotify, Steam, TOR, and World of Warcraft. The included web services are: 4Shared, Amazon, Apple, Ask, Bing, Blogspot, CNN, Craigslist, Cyworld, Doubleclick, eBay, Facebook, Go.com, Google, Instagram, Justin.tv, LinkedIn, Mediafire, MSN, Myspace, Pinterest, Putlocker, QQ.com, Taobao, The Huffington Post, Tumblr, Twitter, Vimeo, VK.com, Wikipedia, Windows Live, Wordpress, Yahoo, and YouTube.

6. We evaluated the ability of several Deep Packet Inspection tools (PACE, OpenDPI, L7-filter, nDPI, Libprotoident, and NBAR) to label flows in order to create datasets serving as a ground-truth for the subsequent evaluation of various traffic classification tools. This contribution is shown in Paper VII. Specifically:

- We showed that PACE achieves the best classification accuracy from all the tested tools, while the best performing open-source classifiers are nDPI and Libprotoident. L7-filter and NBAR, however, should not be considered as reliable tools.
- We described the methodology of testing Cisco NBAR, which is difficult to evaluate, because it works only on Cisco devices. Therefore, we presented how to re-play the collected packets from PCAP files back to the network in a way that the router is able to inspect them. Resolving that issue involved re-writing destination MAC addresses of each packet, as Cisco routers do not support the promiscuous mode. Additionally, the packets needed to be labeled by the identifier of the flow to which they belong. For this purpose, we used the source MAC addresses of the packets, which were re-written in order to contain the encoded flow identifier. We also described how to configure the Cisco router to inspect the traffic by NBAR with Flexible NetFlow, which is able to generate per-flow records. Finally, we showed how to capture NetFlow records from the router, configure the software for inspecting the NetFlow records, and generate a readable output.

7. We designed, developed, and evaluated a multilevel traffic classifier. This contribution is presented in Paper VIII. Specifically:

   - The classification is performed on six levels: *Ethernet*, *IP protocol*, *application*, *behavior*, *content*, and *service provider*.
   - The *Ethernet* and *IP protocol* levels are identified directly based on the corresponding fields from the headers (*EtherType* in Ethernet frames and *Type* in IP packet).
   - The *application* and *behavior* levels are assessed by a statistical classifier based on C5.0 Machine Learning Algorithm.
   - Finally, the *content* and *service provider* levels are identified based on IP addresses.
   - The system is able to deal with unknown traffic, leaving it unclassified on all the levels, instead of assigning the traffic to the most fitting class.
   - The training data for the statistical classifier and the mappings between the different types of content and the IP addresses are created based on the data collected by Volunteer-Based System, while the mappings between the different service providers and the IP addresses are created based on the captured DNS replies.
   - Support for the following applications is built into the system: America's Army, BitTorrent, DHCP, DNS, various file downloaders, eDonkey, FTP, HTTP, HTTPS, NETBIOS, NTP, RDP, RTMP, Skype, SSH, and Telnet.

- Within each application group we identify a number of behaviors – for example, for HTTP, we selected *file transfer*, *web browsing*, *web radio*, and *unknown*.
- Our system built based on the method provides also traffic accounting and it was tested on 2 datasets created by VBS.
- The design and the detailed implementation steps are described.
- Our system was implemented in Java and released as an open-source project.
- Finally, the accuracy of the classification on the application level by our system was compared with the accuracy given by several Deep Packet Inspection tools (PACE, OpenDPI, L7-filter, nDPI, Libprotoident, and NBAR) on the 2 datasets. We showed that our system outperforms other classifiers regarding the applications, which it supports.

8. We developed a method for labeling non-HTTP flows, which belong to web services (as YouTube). Classically, labeling was done based on the corresponding domain names taken from the HTTP header. However, that could allow to identify only the HTTP flows. Other flows (as encrypted SSL / HTTPS flows, RTMP flows) were left unlabeled. Therefore, we implemented a heuristic method for detection of non-HTTP flows, which belong to the specific services. To be recognized as a non-HTTP web flow, the application name associated with the flow should be the name of the web browser (as *chrome*), a name of a web browser plugin (as *plugin-container*, *flashgcplay*), or the name should be missing. Then, we looked at the HTTP flows, which were originated from 2 minutes before to 2 minutes after the non-HTTP web flow. If all the corresponding (originated from the same local machine and reaching the same remote host) HTTP flows had a web service label assigned, and the service label was the same for all of the flows, the non-HTTP flow obtained the same web service label. That allows us to test how the classifiers identify encrypted and other non-HTTP traffic belonging to various web services. This contribution is shown in Paper IX.

9. We directly compared the ability of several DPI tools (PACE, OpenDPI, two configurations of L7-filter, nDPI, Libprotoident, and NBAR) to classify 17 application protocols, 19 applications (also various configurations of the same application), and 34 web services on a dataset of 767 690 labeled flows. This contribution is shown in Paper IX. Specifically:

    - The evaluated application protocols are: DNS, HTTP, ICMP, IMAP (START-TLS and TLS), NETBIOS (name service and session service), SAMBA, NTP, POP3 (plain and TLS), RTMP, SMTP (plain and TLS), SOCKSv5, SSH, and Webdav.

- The evaluated applications (and their configurations) are: 4Shared, America's Army, BitTorrent clients (using plain and encrypted BitTorrent protocol), Dropbox, eDonkey clients (using plain and obfuscated eDonkey protocol), Freenet, FTP clients (in active and passive modes), iTunes, League of Legends, Pando Media Booster, PPLive, PPStream, RDP clients, Skype (including audio conversations, file transfers, video conversations), Sopcast, Spotify, Steam, TOR, and World of Warcraft.
- The evaluated web services are: 4Shared, Amazon, Apple, Ask, Bing, Blogspot, CNN, Craigslist, Cyworld, Doubleclick, eBay, Facebook, Go.com, Google, Instagram, Justin.tv, LinkedIn, Mediafire, MSN, Myspace, Pinterest, Putlocker, QQ.com, Taobao, The Huffington Post, Tumblr, Twitter, Vimeo, VK.com, Wikipedia, Windows Live, Wordpress, Yahoo, and YouTube. We showed that PACE is able to identify the highest number of various web services among all the studied classifiers. PACE detected 16 web services, nDPI 10, OpenDPI 2, L7-filter in its default version only 1, Libprotoident 1, and NBAR none. We have also shown that L7-filter is characterized by a very high number of misclassified flows belonging to web services (usually 80–99 %) – the flows were recognized in a vast majority as *Finger* and *Skype*.
- We evaluated the impact of flow or packet truncation on the detection rate by the particular classifiers. We tested the classifiers on 3 sets of data: containing full flows with entire packets, with truncated packets (the Ethernet frames were overwritten by 0s after the 70th byte), and with truncated flows (we took only 10 first packets for each flow).
- We evaluated the impact of protocol encryption or obfuscation on the detection rate by the particular classifiers. Protocol encryption made the detection rate lower in all the cases, while we did not see such dependency while using obfuscated eDonkey protocol – in this case, PACE demonstrated even increased detection rate from 16.50 % (for plain traffic) to 36.00 %.
- We showed that only PACE is able to identify accurately some applications, which are supposed to be hard to detect, as Freenet or TOR.

10. We showed the design, implementation, and validation of the nDPI traffic classifier. This contribution is presented in Paper X. Specifically:

    - We presented the background and motivations for creating the classifier.
    - nDPI is compared with its predecessor – the OpenDPI library.
    - The current design and implementation of the classifier is described in detail, including the way how the encrypted traffic is handled.
    - After implementing support for new applications, the classifier was validated against 31 popular protocols and applications, and 7 commonly used web

services. We showed that the classifier in the current version is characterized by high accuracy and a very low rate of misclassified flows (for most classes less than 1 %).

- We demonstrated that the test application using nDPI processes packets at an average speed of 3.5 Mpps / 8.85 Gbps using a single core CPU and a PCAP file as the packet source.
- We proposed future enhancements, which make the nDPI classifier consistent. Most of them were already implemented, which resulted in creating nDPIng. More about this project can be read in Section 7.1.

3.1 Bredbånd Nord

Bredbånd Nord A/S[1], a regional fiber networks provider, was our collaboration partner throughout the PhD study. The initial objective of our collaboration was to develop a solution for assessing the Quality of Service and Quality of User Experience in the network belonging to the company. For us, the collaboration benefits included the access to real network data and real production environment, which were supposed to result in the development of an accurate monitoring solution. Our research and the performed measurements were going to be subsequently used to improve the performance of the network and to prove to the users that the quality of their connection is fulfilling the agreement.

The first part of our collaboration was focused on the development of the Volunteer-Based System (VBS). The network users selected by the company agreed to participate in testing the software and volunteered by giving us access to their network data. That allowed us to harden VBS by eliminating the detected bugs and to collect data from various users, which use the network in a different way. These data were used by us in the next phases of the PhD project, for example, as the input for training statistical classifiers based on Machine Learning Algorithms (MLAs), for investigating behaviors of various applications, or as the source of IP addresses associated with different kinds of files transmitted by HTTP. The company obtained a host-based traffic monitoring solution, which could be used for many purposes. The detailed statistics about the network flows give an overview how the network is utilized. That knowledge allows to better allocate the available bandwidth (by creating or improving the Quality of Service policies in the network), tune the network parameters on routers and switches, or to create special facilities for particular application or services (as proxy or cache servers, local DNS servers). Users located in the same network can be compared and assigned to specific profiles, e.g. VoIP users, heavy downloaders, etc. The users can be directly targeted by recommending special Internet packages by their ISPs, as a low-latency one, but also low-throughput connection, or unlimited download rates at night. Assigning

[1] See http://www.bredbaandnord.dk/

these users to separate VLANs allows to control the traffic in a more efficient way, when one user does not interfere with another one.

The second part of our collaboration was focused on the development of a traffic accounting solution, which would be able to provide an overview how the network is utilized. The solution needed to work in a central point in the network, be able to process the traffic at speeds reaching 10 Gbit/s, and was not allowed to compromise users' privacy. We contributed by designing and development of a traffic classifier, which is able to classify the traffic in a consistent manner on all 6 levels: *Ethernet*, *IP protocol*, *application*, *behavior*, *content*, and *service provider*. The *Ethernet* and *IP protocol* levels are identified directly based on the corresponding fields from the headers (*EtherType* in Ethernet frames and *Type* in IP packet). The *application* and *behavior* levels are assessed by a statistical classifier based on C5.0 Machine Learning Algorithm. To train C5.0, we used the data collected by VBS. Finally, the *content* and *service provider* levels are identified based on IP addresses. Bredbånd Nord contributed by giving us access to anonymized one-week traces of the DNS traffic in their network. Thanks to that, we were able to obtain the mappings between the queried domain names and the IP addresses as well as the number of queries for the particular domain. Our system was implemented in Java and released as an open-source project, which is our contribution to the scientific community.

4 State of the Art and Research Questions

This section enumerates and explains the research questions, which needed to be answered in order to achieve the objective. It introduces the relationship between them, our contributions, and the scientific papers included in the thesis. That also positions our work to the state of the art.

4.1 Traffic Classification

Each network application and service has different requirements according to bandwidth, delay, jitter, packet loss, burstiness, and other network parameters. Therefore, in order to assess the quality of the particular session, we need to know what application or service is associated with the examined network flow. To assess the quality in real-time, the involved traffic classification techniques must fulfill some important requirements, which cause numerous questions:

Q: How to classify the traffic in high-speed networks in real-time, maintaining affordable level of allocated resources (CPU, memory usage, bandwidth supported by the network interfaces and the mainboard)?

A: To be classified, the traffic in computer networks must be logically grouped into entities, which represent the packets being transferred between two end points.

The already existing methods for traffic classification usually use for this purpose transport-layer flows, defined as groups of packets, which have the same end IP addresses, ports, and use the same transport layer protocol. Flows can be treated as unidirectional or bidirectional – in the latter case the packets going from the local to the remote host and from the remote to the local host are treated as a part of the same flow.

The first and the easiest solution for the classification of computer traffic is to use transport protocol port numbers [3, 4]. This method is shown to have numerous advantages: it is very fast, low-resource consuming, support for it is implemented by many network devices and it does not inspect the application payload, so it does not compromise users' privacy. However, it is useful only to detect protocols and applications, which use fixed port numbers. P2P applications (including Skype) or FTP data flows cannot be recognized by this method [5–7]. Some applications also can try to use well-known port numbers in order to cheat the security policies in the network.

Deep Packet Inspection (DPI) was designed to address the drawbacks of the classification based on port numbers. Inspecting the actual packet payload makes able to detect applications and services regardless of the port numbers or other values stored in packet headers. However, the existing DPI tools also have numerous drawbacks. At first, they are not able to accurately classify traffic belonging to some applications, as Skype. The reason is that some applications do not have any clear patterns, so the detection is based on the statistical analysis, for example, one of the most commonly used DPI classifier l7-filter (proposed in [8]) is shown to produce a lot of false positives and false negatives [9]. Moreover, the DPI classification is quite slow and requires a lot of processing power [5, 6]. Application signatures for every application must be created outside the system and kept up to date [5], which can be problematic. Encryption techniques in many cases make DPI impossible.

Statistical classification of computer network traffic usually relies on Machine Learning Algorithms, as K-Means, Naive Bayes Filter, C4.5, J48, or Random Forests. They can be used in any point of the network, providing very fast statistical detection of the application, to which the traffic belongs. Achievable detection rate correctness is over 95 % [3–5, 7, 8, 10–12]. It was demonstrated in [5] that all the P2P applications behave similarly, so it is possible to use statistical analysis to detect even unknown applications.

Therefore, we can conclude that according to our studies, statistical methods are the best choice for the real-time classification of traffic in computer networks, while maintaining affordable resource usage.

Q: How to ensure the privacy and confidentiality of the inspected data, and how to conform with the legal terms in the country?

A: We need to avoid performing Deep Packet Inspection in the network as it relies on inspecting the user data and therefore privacy and confidentiality issues can appear [5]. Additionally, DPI is illegal in some countries.

Q: How to perform the classification in a way that it would appear as transparent to the network and its users?

A: A good classification method should be non-visible to the network and its users, which means that it must not influence the performance of the network and the services, which are provided to the users. Again, the statistical classification is the best method regarding these conditions, as it does not slow-down the network, while it provides accurate results.

Q: How to classify the traffic in a consistent manner, i.e. the results should reveal the application, which generated the flow, its content, or a web service provider? However, it is not acceptable to obtain a mix of results on various levels for each flow.

A: All the classification methods described above have one thing in common: they are not able to identify the traffic on multiple levels in a consistent manner. The port-based classification usually gives the name of the application protocol. The results provided by the DPI tools are usually a mix of application names, content names, and service provider names. For some flows only the application is identified (as *HTTP*, *BitTorrent*, or *Skype*), for others only the content (as *audio*, *video*) or content container (as *Flash*), for yet others only the service provider (as *Facebook*, *YouTube*, or *Google*). Machine Learning based tools are not easily able to detect the content carried by the traffic or its service provider.

To overcome the drawbacks of already existing methods, we developed a novel hybrid method shown in Paper VIII to provide accurate identification of computer network traffic on six levels: *Ethernet*, *IP protocol*, *application*, *behavior*, *content*, and *service provider*. The *Ethernet* and *IP protocol* levels are identified directly based on the corresponding fields from the headers (*EtherType* in Ethernet frames and *Type* in IP packet). The *application* and *behavior* levels are assessed by a statistical classifier based on C5.0 Machine Learning Algorithm. Finally, *content* and *service provider* levels are identified based on IP addresses. The training data for the statistical classifier and the mappings between the different types of content and the IP addresses are created based on the data collected by a host-based traffic monitoring system, while the mappings between the different service providers and the IP addresses are created based on the captured DNS replies.

4.2 Machine Learning Algorithms

The previous point led to the conclusion that the traffic classification must be done using the information taken from the packet headers, which is used directly (as Ethernet or IP protocol) or to generate relevant statistics. Statistical classification methods usually involve the usage of Machine Learning Algorithms (MLAs). That creates a new bunch of questions to answer:

Q: Which MLA is the best one for traffic classification?

A: Several tries were made to classify accurately P2P and Skype traffic using older implementations of MLAs, like REPTree, C4.5, or J48. In [5], the authors proposed few simple algorithms based on REPTree and C4.5, which were shown to be able to classify P2P traffic using the first 5 packets of the flow. Their method based on C4.5 performed highly accurately (97 % of P2P traffic was classified properly), but the accuracy was not tested when starting packets from the flow were not in possession.

Another approach to classify P2P applications was taken in [6] using a Java implementation of C4.5 called J48 to distinguish between 5 different applications. The authors tried to skip a number of packets in the beginning of the flow ranging from 10 to 1000 and they obtained only a little fluctuation in performance, with classification accuracy over 96 %. It was shown in [13] that the original C4.5 and J48 perform much different on relatively small or noisy data sets (accuracy of J48 and C5.0 was in tested cases similar, and worse than C4.5). J48 processing using statistics based on sizes was implemented in [14] for detection of BitTorrent and FTP traffic, reaching an accuracy of around 98 %. This publication showed that behavior of data parameters contained in encrypted and unencrypted traffic generated by the same application looks almost the same. Moreover, it was shown that zero-payload packets (ACK) can distort statistics based on sizes.

In [15], different algorithms of classification of the network traffic were evaluated, including C5.0. The achieved accuracy was around 88–97 % on traffic belonging to 14 different application classes. The loss in the classification accuracy was probably partly due to preparing both training and test cases, where the decision attribute (application name) was obtained by DPI tools (PACE, OpenDPI, and L7-filter). These DPI solutions use multiple methods (including statistical analysis) to obtain the application name. Therefore, both training and test data were in some degrees inaccurate, what caused also more errors from the side of C5.0.

Based on the state of the art, we assessed that C5.0 is the most suitable tool for our purposes, as the successor of C4.5 and J48, which are the most common MLAs successfully used for the traffic classification problem.

Q: How should the C5.0 MLA be configured to perform the best?

A: In Paper II, we have tested various configurations of how C5.0 creates the decision model and classifies the subsequent cases: decision trees, rules, boosting, and softening thresholds. The test was performed on a dataset containing 93 572 cases composed of 7 different traffic groups: *Skype*, *FTP*, *BitTorrent*, *web* except web radio, *web radio*, *games*, and *SSH*. The lowest error rate of 0.1 % was achieved by using the boosted classifier, comparing to 0.4 % error rate when using the standard classification without any options. However, creating the boosted classifier took around 10 times more time than creating the standard classifier. Furthermore, our research demonstrated that creating the rules instead of decision trees, or using softened thresholds had no or only a little impact on the error rate, while it extended dramatically the time used for constructing the classifier. Concluding, C5.0 should be left with its default configuration (decision trees), or configured to create a boosted classifier.

Q: How to obtain high-quality training data for C5.0?

A: All the MLAs require a significant amount of training data for the initial learning. The precision of the future classification by MLAs depends heavily on quality of the training data. Most papers show that researchers usually obtain their ground-truth through port-based or DPI-based techniques [12, 16–20]. The poor reliability of port-based techniques is already well known, given the use of dynamic ports or well-known ports of other applications [21–23]. Although the reliability of DPI-based techniques is still unknown, according to conventional wisdom they are, in principle, one of the most accurate techniques.

The use of private datasets is derived from the lack of publicly available datasets with payload. Mainly because of privacy issues, researchers and practitioners are often not allowed to share their datasets with the research community. To the best of our knowledge, just one work has tackled this problem. Gringoli et al. in [24] published anonymized traces without payload, but accurately labeled using his proprietary system called GT.

To address the problem with obtaining a good quality ground-truth, we decided to create our own ground-truth generation system called Volunteer-Based System (VBS), which is described in Paper I.

Q: How many and which classification attributes should we choose?

A: The choice of the classification attributes for computer network traffic is a challenging task, which we have addressed preliminarily in Paper II and in a modified, updated version in Paper VIII. Generally, the selected parameters depend the purpose, for which the traffic classification is performed. If it is done for a real-time QoS monitoring (as in Paper II), we can inspect only parts of some selected flows

from the network, which gives us other possibilities and challenges than traffic classification for data accounting, which was shown in Paper VIII. However, in both cases, we decided to rely on the parameters based on packet sizes, while avoiding the usage of any time-based parameters (as flow duration or frame inter-arrival time). The reason was that we assumed that flow characteristics based on packet sizes within a network are independent of current conditions, contrary to the flow characteristics based on time parameters (which change quickly during e.g. congestion). Other attributes contain the transport layer protocol name, remote port numbers, number of packets having selected set of TCP flags, or number of packets without any payload in the sample. The number of the attributes for traffic classification is not so important as for clustering techniques, when it must be kept on a relatively low level, as each new attribute prolongs the time of constructing the cluster in an exponential way. However, it is important to avoid including attributes that contain values specific only for a small number of cases belonging to a particular class. That can result in overtraining the classifier – it will be extremely good at recognizing the class of the training cases, while it will not be able to identify a yet unknown case.

Q: How should the selected classification attributes be calculated for each flow? Do we need to possess the entire flow, or is it enough to have just a number of packets? If yes, how many packets, and from which part of the flow?

A: As shown in Paper II, it is enough to calculate the classification attributes for a sample containing 5 consecutive packets to achieve the classification accuracy of 95 % or more. The test was performed on a dataset containing cases composed of 7 different traffic groups: *Skype*, *FTP*, *BitTorrent*, *web* except web radio, *web radio*, *games*, and *SSH*. However, we showed that the accuracy can be significantly improved (to above 99 %), when the statistics are calculated based on 35 consecutive packets. Further increasing of the number of packets in the sample does not increase the classification accuracy. It is worth mentioning that first few packets in each flow can have different characteristics than the rest, so we decided to skip first 10 packets in each flow from being included in the sample.

Q: How to handle flows, which belong to other categories than the ones selected by us and trained by MLAs?

A: Generally, classifiers based on MLAs do not have any possibility to recognize the class of the elements, if the class was not present among the training data. Furthermore, such classifiers cannot leave any case as unknown – each case is processed according to the decision tree and assigned to a class. A good example can be a situation, when among the training data we have only 2 classes: *DNS* and *HTTP*, and among 50 different classification attributes we have the transport protocol name. Then, probably the classifier is going to construct the decision

tree in a very simple way: All the cases, which are associated with TCP will be assigned to the *HTTP* class, and all the cases, which are associated with UDP will be assigned to the *DNS* class – in both cases regardless of other classification attributes. If the generated decision tree will be used to classify a case originated from BitTorrent traffic, it is going to be assigned to either *DNS* or *HTTP* class, depending on the used transport layer protocol.

This problem was addressed by us in Paper VIII, Section 3.1 on page 178. The solution was to include selected cases, which do not fall into any of the tested traffic classes, to the *UNKNOWN* class. Then, the classifier is able to distinguish other kinds of traffic from the groups of traffic defined by us. The procedure of selecting the proper cases to be included in the *UNKNOWN* class is described in detail in the paper referenced above.

Q: Can we use MLAs to recognize the content transmitted by the flow, its behavior, or the web service provider (for web flows)?

A: Normally, Ethernet, IP and TCP/UDP packet headers do not contain any information about the content (as audio, video, etc) transported by the flow, or about the web service provider (as Yahoo, Facebook), with which the flow is associated. In some cases, these information can be extracted from HTTP headers, which can be considered as a DPI technique. Therefore, MLAs are not the best solution for determining such flow characteristics, but they can be recognized with high probability based on the IP addresses – an example of a such solution is shown in Paper VIII. However, the behavior of a flow (as streaming, web browsing, bulk download) influences directly its classification arguments, as packet sizes, distribution of packets without payload or with certain TCP flags, upload to download ratio, etc. So, in this case, MLAs do a good job.

4.3 Ground-Truth for MLAs

We emphasized that it is almost impossible to use already pre-classified data (as public network traces) as the training data for MLAs, nor use DPI tools or other classification tools to pre-classify the data generated by us. Therefore, we were challenged by many new questions:

Q: How to label the collected network flows, so they can be used as the training data for MLAs?

A: A good solution for obtaining accurate training data can rely on collecting the flows at the user side along with the name of the associated application. We did that by using our Volunteer-Based System (VBS). The basic idea and design of the system was described in [1] and our current implementation in Paper I. The

system consists of clients installed on users' computers, and a server responsible for storing the collected data. The task of the client is to register the information about each flow passing the Network Interface Card (NIC), with the exception of traffic to and from the local subnet, to prevent capturing transfers between local peers.

Q: Which information should we collect about the network flows?

A: In order to calculate the statistics based on the header information and packet sizes, we need to collect both the information, which is common for the flow, and the information, which is specific for each packet. In case of our VBS, the following attributes of the flow are captured: start and end time of the flow, number of packets contained by the flow, local and remote IP addresses, local and remote ports, transport layer protocol, name of the application, and identifier of the client associated with the flow. The client also collects information about all the packets associated with each flow: direction, size, TCP flags, and relative timestamp to the previous packet in the flow. One transport-layer flow can contain multiple application-layer streams of HTTP data, and each of them can carry different kinds of content, such as audio or video. For that reason, packets belonging to flows which carry HTTP content require additional information to be collected. Therefore, in this case, we additionally store the information about the content type for each packet of the flow. In fact, the information about the content type is present only in the first packet of the response made to an HTTP request. It means that for each HTTP request we have one packet containing the information about the content type, which allows us to logically split all the application-layer HTTP streams.

Q: How to protect the users' privacy?

A: Our VBS does not perform any type of DPI except the inspection of HTTP headers to extract the value of the *content-type* field. All the other stored information originates directly from IP or TCP/UDP headers or system sockets. To enforce the users' privacy, the IP addresses are processed by one-way hash function before they are transmitted from the user's computer to the server.

Q: Can we say anything about the content carried by the flows or the web service providers? If yes, how can we use this information in our research?

A: We only can recognize the content transmitted by HTTP due to the presence of the *content-type* field in HTTP headers.

Q: How can we distinguish between different kinds of data transmitted by the same application?

A: Well, it is possible only in some cases. At first, when the applications uses HTTP to transfer different types of data. At second, when the application uses HTTP together with another protocol, so we can at least separate HTTP. At third, we can use the port numbers, if we know that the selected application uses the concrete port numbers for the selected things.

4.4 DPI Tools in Ground-Truth Establishment

Many researchers use various DPI tools to pre-classify their data, which become the input to train and test MLAs. As high-quality training data is crucial for our MLA-based classification system, we needed to evaluate the use of DPI tools as a common ground-truth establishment technique and answer several important questions:

Q: Which DPI tools are the best for establishing the ground-truth?

A: Some works tried to evaluate the accuracy of DPI-based techniques [18, 20, 25]. However, their results rely on a ground-truth generated by another DPI-based tool [20], port-based technique [18], or a methodology of unknown reliability [25]. To the best of our knowledge, only [26] addressed this problem. Using a dataset obtained by GT [24], the reliability of L7-filter and a port-based technique was compared, showing that both techniques present severe problems to accurately classify all the traffic.

In Paper VII, we have compared the performance of six DPI tools (PACE, OpenDPI, L7-filter, nDPI, Libprotoident, and NBAR), which are usually used for the traffic classification. In order to make the study trustworthy, we have created a dataset using VBS [27]. This tool associates the name of the process to each flow making its labeling totally reliable. The dataset of more than 500 K flows contains traffic from popular applications like HTTP, eDonkey, BitTorrent, FTP, DNS, NTP, RDP, NETBIOS, SSH, and RDP. The total amount of data properly labeled is 32.61 GB. We found that *PACE* is the best classifier. Even while we were not using the last version of the software, PACE was able to properly classify 94 % of our dataset. Surprisingly for us, *Libprotoident* achieves similar results, although this tool only inspect the first four bytes of payload for each direction. On the other hand, *L7-filter* and *NBAR* perform poorly in classifying the traffic from our dataset.

Q: Considering the accuracy and the consistence of the given output by the DPI tools, can we use them as an alternative to VBS?

A: As shown in Paper VIII, Section 6 (page 192), the most accurate classifiers (PACE, nDPI, and Libprotoident) do not provide any consistent output – the results are given on a mix of various levels: application, content, content container, service

provider, or transport layer protocol. On the other hand, L7-filter and NBAR provide results consistently on the application level, however, their accuracy is too low to consider them as tools for generating the ground-truth. Therefore, the currently available DPI tools are not an alternative for systems as VBS.

Q: How can we test the accuracy of Cisco Network-Based Application Recognition (NBAR), which runs on Cisco hardware?

A: Preparing the data for NBAR classification is more complicated, since no information describing the flows (as flow identifiers, start and end timestamps) can be provided to the router. Therefore, the packets need to be extracted in a way that allows the router to process and correctly group them into flows and label the flows, so the output can be associated with the original flow. We achieved that by changing both the source and destination MAC addresses during the extraction process. The destination MAC address of every packet must match up with the MAC address of the interface of the router, because the router cannot process any packet which is not directed to its interface on the MAC layer. The source MAC address was set up to contain the identifier of the flow to which it belongs, so the flows were recognized by the router according to our demands.

Classification by NBAR required us to set up a full working environment. We used GNS3 – a graphical framework, which uses Dynamips to emulate our Cisco hardware. We emulated the 7200 platform, since only for this platform supported by GNS3 was available the newest version of Cisco IOS (version 15), which contains Flexible NetFlow. The router was configured by us to use Flexible NetFlow with NBAR on the created interface. Flexible NetFlow was set up to create the flows taking into account the same parameters as are used to create the flow by VBS. On the computer, we used *tcpreplay* to replay the PCAP files to the router with the maximal speed, which did not cause packet loss. At the same time, we used *nfacctd*, which is a part of PMACCT tools, to capture the Flexible NetFlow records sent by the router to the computer. The records, which contain the flow identifier (encoded as source MAC address) and the name of the application recognized by NBAR, were saved into text log files. This process is broadly elaborated in our technical report [28].

To the best of our knowledge, our Paper VII is the first work to present a scientific performance evaluation of NBAR.

4.5 DPI Tools in Direct Traffic Classification and Accounting

As we showed in the previous paragraphs, using MLAs in traffic classification and accounting has many advantages, including high speed, low resource usage, and no privacy concerns. However, many traffic classification and accounting system rely on DPI, so we also tried to evaluate this approach. That raised several new questions, which were

not covered by the general assessment of traffic classification tools in the beginning of this section:

Q: Which DPI tool is most frequently evaluated in the scientific literature?

A: OpenDPI amounts for most of the publications [18, 25, 29–32].

Q: What is the accuracy of OpenDPI according to the scientific literature?

A: According to [29], the test performed by the European Networking Tester Center (EANTC) in 2009 resulted in 99 % of detection and accuracy for popular P2P protocols by OpenDPI. The big amount of flows marked as *unknown* by OpenDPI was confirmed in [30], where the authors made an effort to calculate various parameters for traffic originated from different applications: number of flows, data volume, flow sizes, number of concurrent flows, and inter-arrival times. The study was based on 3.297 TB of packets collected during 14 days from an access network for around 600 households. 80.1 % of the flows, amounting for 64 % of the traffic volume, were marked as *unknown* by OpenDPI.

In [29], the authors study the impact of per-packet payload sampling (i.e., packet truncation) and per-flow packet sampling (i.e., collect only the first packets of a flow) on the performance of OpenDPI. The results show that OpenDPI is able to keep the accuracy higher than 90-99% with only the first 4-10 packets of a flow. The impact by the per-packet payload sampling is considerably higher. Their results use as ground-truth the dataset labeled by OpenDPI with no sampling. Thus, the actual classification of the dataset is unknown and no possible comparison with our work can be done.

Similar work, performed by the same authors, is described in [31]. The goal was to find out what is the suggested number of packets from each flow, which needs to be inspected by OpenDPI in order to achieve good accuracy, while maintaining a low computational cost. The focus was on Peer-to-Peer (P2P) protocols. The test was performed on a 3 GB randomly selected subset of flows from the data collected at an access link of an institution over 3 days. The authors found that inspecting only 10 packets from each flow lowered the classification abilities of P2P flows by OpenDPI by just 0.85 % comparing to the classification of full flows, while saving more than 9 % of time.

Q: Are there any studies regarding the accuracy of L7-filter?

A: In [25], the authors tested the accuracy of L7-filter and OpenDPI, and they also built their own version of L7-filter with enhanced abilities of classification of the UDP traffic. The data used in the experiment were collected by Wireshark, while the applications were running in the background. The data were split into 27 traces, each for one application, where all the applications were supported by

both L7-filter and OpenDPI. Other flows were removed from the dataset. However, they do not explain how they validate the process of the isolation of the different applications. The obtained precision was 100 % in all the cases (none of the classification tools gave a false positive), while the recall deviated from 67 % for the standard L7-filter, through 74 % for their own implementation of L7-filter, and 87 % for OpenDPI.

Q: What is the accuracy of Libprotoident according to the scientific literature?

A: In [20], the developers of Libprotoident evaluated the accuracy of the classification of this tool and compared the results with OpenDPI, Nmap, and L7-filter. The ground-truth was established by PACE, so only the flows recognized by PACE were taken into account during the experiment. The accuracy was tested on two datasets: one taken from the Auckland university network, and one from an Internet Service Provider (ISP). On the first dataset, Libprotoident had the lowest error rate of less than 1 % (OpenDPI: 1.5 %, L7-filter: 12.3 %, Nmap: 48 %.). On the second dataset, Libprotoident achieved the error rate of 13.7 %, while OpenDPI 23.3 %, L7-filter 22 %, and Nmap 68.9 %. The authors claim that Libprotoident identified 65 % of BitTorrent traffic and nearly 100 % of HTTP, SMTP, and SSL.

Q: What is the accuracy of NBAR according to the scientific literature?

A: To the best of our knowledge there are no accessible research studies or reports about the accuracy of NBAR. However, an experiment was made to assess how big amount of network traffic is classified by NBAR and L7-filter, and how big amount of traffic is left as *unknown* [33]. The authors captured by Wireshark all the packets flowing in a local network of an IT company during 1 hour. From 27 502 observed packets, 12.56 % were reported as unknown by NBAR, and 30.44 % were reported as unknown by L7-filter.

Q: Where can I find even a broader evaluation of different DPI tools?

A: A very comprehensive review of different methods for traffic classification was made in 2013 by Silvio Valenti et al. [16]. The authors refer to 68 different positions in the literature and cover the topic from the basis to more advanced topics, mostly dealing with Machine Learning Algorithms (MLAs).

Q: Can we use DPI tools for traffic accounting purposes?

A: As we mentioned before, the evaluated by us DPI tools either give consistent results on the application level of poor accuracy (L7-filter and NBAR) or provide much better results in terms of accuracy, but not consistent on any level (PACE, OpenDPI, nDPI, and Libprotoident). Traffic accounting relying on the classification by tools characterized by poor accuracy is obviously highly discouraged. The

inconsistent results obtained from the remaining tools can be in some situations acceptable, namely, when only a general view of the network usage is required. However, the already discussed DPI classifiers are not able to deliver results, which make us able to see what is the network usage by particular transport-layer protocols (TCP, UDP), application-layer protocols (as SMTP, HTTP, or BitTorrent), content (as audio or video), or web services (as Facebook or YouTube), as the mix of results on different levels is returned. Therefore, the usefulness of the DPI tools for traffic accounting purposes is limited to simple overview scenarios. However, in Paper X, we proposed future enhancements, which make the nDPI classifier consistent. Most of them were already implemented, which resulted in creating nDPIng. More about this new project can be read in Section 7.1.

Q: Can we use DPI tools to recognize encrypted traffic from various applications?

A: Generally yes. In Paper IX, we performed an evaluation of the DPI tools (PACE, OpenDPI, L7-filter, nDPI, Libprotoident, and NBAR) in recognizing several different encrypted application protocols (as, IMAP-STARTTLS, IMAP-TLS, POP3-TLS, SMTP-TLS, Webdav, and BitTorrent) and web services, which use SSL connections (as Facebook or Google). Flows representing IMAP-STARTTLS were detected 100 % correctly, while IMAP-TLS was detected only by Libprotoident and NBAR (the rest of the tools recognized these flows as regular SSL flows). POP3-TLS and SMTP-TLS were detected by Libprotoident on 100 % of flows, while the other tools recognized these flows as regular SSL flows. Some flows representing Webdav (3.51 %) were detected only by PACE.

Encrypted BitTorrent flows were identified most accurately by PACE, which was able to recognize 78.68 % of them. Libprotoident was able to identify 60.31 % of flows, nDPI 54.41 %, L7-filter around 40.50 % depending on the version, NBAR 1.29 %, while OpenDPI only 0.27 %. The unencrypted BitTorrent flows were very well classified by all the tools. The best detection rate had PACE (99.87 %), nDPI (99.41 %), Libprotoident (99.30 %), while the worst had NBAR (77.84 %).

Q: Are DPI tools capable of detecting protocols, which use obfuscation?

A: In Paper IX, we studied the ability of various DPI classifiers to identify the obfuscated eDonkey protocol and we compared the results with the results from evaluation of the plain eDonkey protocol. The best detection rate of the obfuscated flows had PACE (36.06 %). Libprotoident, nDPI, and L7-filter could detect around 11 % of these flows. Other classifiers did not detect this protocol at all. The highest rate of misclassifications we obtained from L7-filter (16.59 %) as *Finger*, *RTP*, and *Skype*. However, the plain eDonkey protocol also was difficult to identify. PACE achieved the detection rate of 16.50 %, which was over 2 times lower than regarding the obfuscated version of this protocol. Libprotoident and

L7-filter also detected around 18 % of flows, nDPI around 15.50 %, while the other classifiers did not maintain this ability (the number of correctly classified flows did not exceed 3 %). The biggest number of wrong classifications we obtained from L7-filter (16.32 %) as *Finger*, *RTP*, and *Skype*.

Q: How the DPI tools perform in recognizing applications, which are intentionally difficult to identify?

A: There are several applications (as Freenet or TOR), which were created in order to provide anonymity to their users and to allow the users to access the network resources, which are not available in the particular place. That concerns, for example, several applications and web services banned in China. Therefore, the traffic generated by these applications is suspected to be difficult to identify as belonging to these applications. In Paper IX, we assessed the ability of various DPI tools to classify the traffic generated by these two applications. Freenet was recognized only by PACE (79.26 % of flows). The rest of the classifiers did not identify this traffic and we observed a number of misclassifications: 20.00 % as *Finger*, *Skype*, and *NTP* by L7-filter, and 15.56 % as *RTP* by NBAR. Regarding TOR traffic, it was recognized accurately by PACE (85.95 % of flows). nDPI and Libprotoident detected 33.51 % of flows, while the other tools were not able to identify the traffic generated by TOR. The unclassified flows were usually marked as *SSL* or *HTTPS*.

Q: Which DPI tools are the best in recognizing the web services?

A: In Paper IX, we performed the evaluation of various DPI tools regarding their ability to discover flows belonging to 34 different web services. Specifically:

- PACE recognized 4Shared (84.69 %), Amazon (58.97 %), Apple (0.84 %), Blogspot (3.83 %), eBay (67.97 %), Facebook (80.79 %), Google (10.79 %), Instagram (88.89 %), Linkedin (77.42 %), Mediafire (30.30 %), Myspace (100 %), QQ.com (32.14 %), Twitter (71.18 %), Windows Live (96.15 %), Yahoo (54.70 %), and YouTube (81.97 %). PACE did not have problems with recognizing SSL flows belonging to these services, which means that PACE must use other techniques than just looking directly into the packets to associate the flows with the particular services. PACE was the classifier, which was able to identify the highest number of various web services.
- OpenDPI recognized only Direct Download websites: 4Shared (83.67 %) and MediaFire (30.30 %).
- L7-filter recognized only Apple (0.42 %). L7-filter (especially L7-filter-all) is characterized by a very high number of misclassified flows belonging to web services (usually 80–99 %). The flows were recognized in a vast majority as Finger and Skype.

- nDPI recognized Amazon (83.89 %), Apple (74.63 %), Blogspot (4.68 %), Doubleclick (85.92 %), eBay (72.24 %), Facebook (80.14 %), Google (82.39 %), Yahoo (83.16 %), Wikipedia (68.96 %), and YouTube (82.16 %) being the second best technique on this level.
- Libprotoident recognized only Yahoo (2.36 %). This result is understandable given that Libprotoident only uses the first 4 bytes of packet payload to classify a flow, making considerably more difficult a specific classification as web service.
- NBAR did not recognize any web services.

Q: What is the impact of packet or flow truncation on the traffic classification?

A: We showed in Paper IX that the impact or flow or packet truncation is highly dependent on the classifier. We tested the classifiers on 3 sets of data: containing full flows with entire packets, with truncated packets (the Ethernet frames were overwritten by 0s after the 70th byte), and with truncated flows (we took only 10 first packets for each flow).

Truncation of packets has a considerable impact on the classification of most *application protocols* by all tools except Libprotoident and NBAR, which tend to maintain their normal accuracy. This suggests that NBAR can be somehow implemented as Libprotoident to classify *application protocols* while the rest of techniques base their classification on the complete flow. L7-filter is not able to detect DNS traffic on this set, while all the other classifiers present the accuracy of over 99 %. Unexpectedly, NBAR cannot detect NTP on the normal set, while it detects it 100 % correctly on the set with truncated packets. We can not present a verifiable reason of this result given that NBAR is not an open-source tool. On the *application* level, only Libprotoident is able to keep its normal accuracy whereas the rest of techniques considerably decreases their accuracies. Regarding the *web services* level, only nDPI is able to recognize some *web services* in this set. Exceptionally, the detection rate is almost the same as for the normal set. Other classifiers tend to leave such traffic as *unknown*. Furthermore, the only tool, which was able to detect web services on the set with truncated packets was nDPI, so we believe that it must use another method for matching the flows to the services than looking for the domain name in the payload.

Truncation of flows does not have any noticeable impact on the classification of *application protocols*. This result suggests that the classification of *application protocols* relies on patterns or signatures extracted from the first packets of the flows. Similar behavior is obtained on the *application* level. However, in this case the impact on the classification of *applications* is noticeable – the detection rate decreases. The only exception is Libprotoident, which provides the same results as for the normal dataset. Therefore, this insinuate that the classification of

some *applications* probably rely on techniques based on statistics (e.g., Machine Learning). FTP in the active mode is a very interesting case, as Libprotoident maintains its 100 % accuracy, while the accuracy of the other classifiers drops to 5.56 %. An strange case is presented with Plain eDonkey traffic, as the best classification accuracy (45.28 %) we obtained by using PACE on the set with truncated flows, while the accuracy on the normal set was only 16.50 %. The percentage of correctly classified *web services* is usually the same or nearly the same as for the normal set.

4.6 QoS Assessment

Finally, we needed to consider some questions regarding the QoS assessment:

Q: How to monitor high-speed networks in order to assess the present QoS?

A: During the last 20 years, we have been witnesses to the subsequent and increasing growth of the global Internet and the network technology in general. The broadband and mobile broadband performance today is mainly measured and monitored by speed. However, there are several other parameters, which are important for critical business and real-time applications, such as voice and video applications or first-person shooter games. These parameters include round trip time, jitter, packet loss, and availability [34, 35].

The lack of the centralized administration makes it difficult to impose a common measurement infrastructure or protocol. For example, the deployment of active testing devices throughout the Internet would require a separate arrangement with each service provider [34]. This state of affairs led to some attempts to make simulation systems representing real characteristics of the traffic in the network. Routers and traffic analyzers provide passive single-point measurements. They do not measure performance directly, but traffic characteristics are strongly correlated with performance. Routers and switches usually feature a capability to mirror incoming traffic to a specific port, where a traffic meter can be attached. The main difficulty in passive traffic monitoring is the steadily increasing rate of transmission links (10 or 100 GB/s), which can simply overwhelm routers or traffic analyzers, which try to process packets. It forces the introduction of packet sampling techniques and, therefore, it also introduces the possibility of inaccuracies. Even at 1 Gbit/s, the measurement can result in enormous amount of data to process and store within the monitoring period [34].

To overcome the heavy load in the backbone and to not introduce inaccuracies, a smart monitoring algorithm was needed. There are several approaches to estimate which traffic flows need to be sampled. Path anomaly detection algorithm was proposed in [36]. The objective was to identify the paths, whose delay exceeds

their threshold, without calculating delays for all paths. Path anomalies are typically rare events, and for the most part, the system operates normally, so there is no need to continuously compute delays for all the paths, wasting processor, memory, and storage resources [36]. Authors propose a sampling-based heuristic to compute a small set of paths to monitor, reducing monitoring overhead by nearly 50 % comparing to monitoring all the existing paths. The next proposals on how to sample network traffic in an efficient way were made on the basis of adaptive statistical sampling techniques, and they are presented in [37] and [38].

Each network carries data for numerous different kinds of applications. QoS requirements are dependent on the service. The main service-specific parameters are bandwidth, delay, jitter, and packet loss. Regarding delay, we can distinguish strict real time constraints for voice and video conferences, and interactive services from delivery in relaxed time frame. In a conversation, a delay of about 100 ms is hardly noticeable, but 250 ms delay means an essential degradation of the transmission quality, and more than 400 ms is considered as severely disturbing [39]. Therefore, in order to provide detailed information about the quality level for the given service in the network, we need to know, what kind of data is flowing in the network at the present time. Processing all the packets flowing in a high-speed network and examining their payload to get the application name is a very hard task, involving large amounts of processing power and storage capacity. Furthermore, numerous privacy and confidentiality issues can arise. A solution for this problem can be the use of Machine Learning Algorithms (MLAs), which rely on previously generated decision rules based on some statistical information about the traffic. In our research, we used one of the newest MLAs - C5.0. MLAs need very precise training sets to learn how to accurately classify the data, so the first issue to be solved was to find a way to collect high-quality training statistics. The QoS assessment method was described in Paper III.

Q: How can we detect the location of the problem in the network?

A: Detecting the location of a congestion is a challenging problem due to several reasons. First of all, we cannot send many probing packets, because it causes too much overhead, and it even expands the congestion. Secondly, without router support, the only related signals to the end applications are packet losses and delays. If packet losses were completely synchronized (packets were dropped from all the flows), the problem would be trivial. In the reality, the packet loss pattern is only partially synchronized [40]. Authors of [40] attempted to solve the problem of detecting the location of the congestion by using the synchronization of the behavior of loss and delay across multiple TCP sessions in the area controlled by the same local gateway. If many flows see synchronized congestion, the local link is the congested link. If the congested link is remote, it is less likely that many flows from the same host pass the same congested link at the same time. If there

is only a small number of flows which see the congestion, the authors performed an algorithm based on queuing delay patterns. If the local link is congested, most flows typically experience high delays at a similar level. Otherwise, the congestion is remote [40].

Q: What are the strengths and weaknesses of using active or passive methods for QoS monitoring?

A: Passive QoS measurements methods can rely on the protocol composition. Usually, predominance of the TCP traffic is observed (around 95 % of the traffic mix) [34]. When congestion occurs, TCP sources respond by reducing their offered load, whereas UDP sources do not. It results in the higher ratio of UDP to TCP traffic. If the proportion becomes high and the bandwidth available to TCP connections becomes too low to maintain a reasonable transmission window, the packet loss increases dramatically (and TCP flows become dominated by retransmission timeouts) [34]. Packet sizes provide insight into the type of packet, e.g. short 40–44 bytes packets are usually TCP acknowledgment or TCP control segments (SYN, FIN, or RST) [34].

Active methods for QoS monitoring raise three major concerns. First, the introduction of the test traffic will increase the network load, which can be viewed as an overhead cost for active methods. Second, the test traffic can affect measurements. Third, the traffic entering ISP can be considered as invasive and discarded or assigned to a low-priority class [34]. Within an administrative domain (but not across the entire Internet), performance can be actively monitored using the datalink layer protocol below IP, as the Operations, Administration and Maintenance (OAM) procedure in ATM and MPLS networks. As a result, at the IP layer it is often desirable to measure performance using the IP/ICMP protocol. So far, most tools or methods are based on ping (ICMP echo request and echo reply messages) or traceroute (which exploits the TTL field in the header of the IP packet) [34].

Although the round-trip times measured by ping are important, ping is unable to measure the one-way delay without additional means like GPS to synchronize clocks at the source and destination hosts. Another difficulty is that pings are often discarded or low-prioritized in many ISP networks. Traceroute will not encounter this problem because UDP packets are used. However, traceroute has known limitations. For example, successive UDP packets sent by traceroute are not guaranteed to follow the same path. Also, a returned ICMP message may not follow the same path as the UDP packet that triggered it [34]. The end-to-end performance measurements can be carried out at the IP layer or the transport/application layer, but the latest is capable of measurements closer to user's perspective. The basic idea is to run a program emulating a particular application that will send traffic through the Internet. All the parameters (delay, loss, throughput,

etc) are measured on the test traffic. This approach has one major drawback - a custom software needs to be installed at the measurement hosts [34].

On the basis of the mentioned work, we found out that the existing solutions are not sufficient for precise QoS measurements. This state of affairs motivated us to create a new system shown in Paper III, which combines both passive and active measurement technologies. The passive mode relies mostly on time-based statistics, which are obtained directly from the flow passing the measurement point. This way, we can assess the jitter, the burstiness and the transmission speed (both download and upload). Unfortunately, it is not possible to receive information about the packet loss or the delay for other than TCP streams while using this method. For that reason, additional tools performing active measurements must be involved in the process of estimating the QoS. One option is to use the ping-based approach, as it can measure both delay and packet loss. Unfortunately, other issues can arise. Ping requests and responses are often blocked by network administrator, or their priority is modified (decreased to save the bandwidth or increased to cheat the users about the quality of the connection). Other options include sending IP packets with various TTL and awaiting *Time Exceeded* ICMP messages, which are usually allowed to be transmitted in all the networks and their priority is not changed. Active measurements must be done in both directions (from the user and from the remote side). The total packet loss and the delay can be calculated as the sum of the delays and the packet losses from both directions of the flow. Furthermore, the knowledge of the direction that causes problems can be used to assess if the problems are located in the local network or somewhere outside.

Q: Can we use the MLA trained in one network to classify the traffic in another one?

A: It was found in [4] that the results of the classification are most accurate when the classifier was trained in the same network as the classification process was performed. This may be due to different parameters, which are constant in the particular network, but which differ among various networks. A good example is the Maximum Transmission Unit, which can easily influence statistics based on sizes. Therefore, in our design, we decided to train the C5.0-based classifier by volunteers in the same network as the classifier will be installed. This allows us to make a self-learning system, where a group of volunteers in the network delivers data used for training the classifier constantly improving its accuracy, while all the users can be monitored in the core using the generated decision rules. The next advantage of the design is that even if some network users cannot participate in the data collecting process because of using other operating systems or devices than supported (like MacOS, Apple or Android smartphones), they will still be able to be monitored in the core of the network because of the rules created on the basis of data collected from the other users.

5 Summary of Papers

This section summarizes the eleven papers of the thesis. Each paper is briefly described, addressing its position in the thesis as well as its contributions and limitations.

Paper I – Volunteer-Based System for Research on the Internet Traffic

This paper presents a new system for host-based traffic monitoring. The system was created in order to perform the traffic analysis on a per-application basis. The current means of traffic classification on the application layer were shown to not be sufficient to deliver accurate data for constructing statistical classifiers based on Machine Learning Algorithms. Our system relies on volunteers who install the client software, which inspects all the traffic passing through the network interface card. Therefore, our software was called Volunteer-Based System (VBS). VBS groups the packets in flows based on 5-tuple: local and remote IP addresses, local and remote ports, and the transport-layer protocol. For each flow the following attributes are logged: the identifier of the client, start time, end time, local IP, remote IP, local port, remote port, transport protocol name, and the application name taken from system sockets. Additionally, some information are logged about each packet in the flow: the flow identifier, packet size, state of all TCP flags, and relative timestamp to the previous packet in the flow. The system was equipped with the ability to process HTTP flows and split them into parts, which carry different files. One HTTP flow can transfer lots of files: HTML, JavaScript, CSS files, various images, audio, video, and other binary files. Because the network characteristics of each of the part can be different, it is crucial to be able to separate them. This makes able to create transfer profiles of different types of contents by HTTP. Additionally, a lot of statistical information can be obtained: which types of files are most commonly transported by HTTP, what is their size, packet size distribution, transfer time. All the statistical information are transmitted to the server located at Aalborg University, where they are stored in a MySQL database. The collected data can be used to create realistic models of traffic originated by different applications, perform various kinds of simulations, and to obtain accurate statistics for training Machine Learning based traffic classifiers.

The paper shows the detailed design and implementation of VBS in Java, so the reader is able to re-construct the system. The final version of VBS was published on SourceForge as an open-source project. A fully automatic update system was implemented in order to facilitate zero-touch updates, transparent to the users. The performance of VBS clients was assessed on 27 computers located in private houses as well as educational institutions in Denmark and Poland. The results showed that the CPU usage observed during 3 months was around 5 % in average, while the temporary consumption of over 10 % is extremely rare. The average RAM usage was around 5 %. It

was shown that around 12 % of the flows were collected without the associated application name. Our system relies on external tools checking the system sockets every 1 second, so the socket for some short flows may not be noticed. In our testing case, the flows without an associated application name contained 5 packets in average, while the flows with the associated application name contained 49 packets in average. The system was shown to have other limitations as well. VBS cannot distinguish different types of flows generated by the same application, for example, different kinds of web browser flows. The same considers tunneled traffic (SSH, CIFS, SAMBA, etc.). Also, for now only IPv4 is supported.

Finally, a thorough threat assessment of the system was performed. It was shown that only the SQL injection attack should be taken into account while designing the future security policies, while the other threats can be left out.

Paper II – A Method for Classification of Network Traffic Based on C5.0 Machine Learning Algorithm

This paper is the first step to the real-time Quality of Service assessment described in Paper III. VBS was used to label flows originated from 7 different groups: *Skype*, *FTP*, *BitTorrent*, *web* except web radio, *web radio*, *games*, and *SSH*. The labeled flows were split into two disjoint sets: one of them was used to generate the statistics for training C5.0 classifier, and the second one was used to assess the accuracy of the classification. We tried to determine how many packets from a flow are needed to construct accurate statistics and if it matters if we have captured the whole flow. The flows were split into chunks of 5–90 packets, from which we generated the statistics. It was shown that the highest accuracy (oscillating around 99.8 %) we obtained if we took the statistics obtained from the chunks created from at least 35 consecutive packets from a random point of the flow, except the first 10 packets from a flow, which can have different characteristic than the rest. Several different modes of C5.0 tools were tested: normal decision-tree based classification, classification based on rules, boosting, and softening thresholds. Boosting was shown to improve the classification accuracy in a small, but still measurable manner by around 0.1–0.5 %, while other modes were characterized by a comparable accuracy. Boosting, however, slower down the process of creating the classifier around 8 times. The relevance of various classification attributes was tested as well. We tested the classifier by using at first 32 general attributes based on packet sizes, then on a full dataset of 42 attributes, where the additional 10 attributes were protocol-dependent (as transport protocol, number of TCP flags). It was shown that by using the full dataset, we lowered the error rate by approximately 15 times. While the final average accuracy of classification of the traffic belonging to the 7 applications was above 99 %, we had a significant number of misclassifications between the FTP and BitTorrent groups. That was a consequence of using mainly size-based classification attributes, which are similar for both types of file transfer.

Paper III – A Method for Evaluation of Quality of Service in Computer Networks

The quality of the network connection experienced by the user is shown to be service-dependent. The generally used network parameters (as bandwidth, delay, jitter, packet loss) are expected to have different values for different kinds of network activity. For example, a voice conversation expects a low delay and jitter, while the available bandwidth is not an issue. However, file transfers expect the highest possible bandwidth, while the delay and jitter are not a concern. Therefore, assessing the network QoS requires the information what kind of traffic is flowing in the network at the particular time. Classification of traffic in high-speed networks is shown to be a demanding task. While port-based classifiers are nowadays ineffective, Deep Packet Inspection requires a lot of processing resources, involves a lot of privacy and law concerns, and is not feasible for encrypted or tunneled protocols. Therefore, the solution is to use statistical classifiers based on Machine Learning Algorithms (MLAs), which require good quality training data.

The paper contains a thorough description of the current methods for obtaining pre-classified traffic, which can be used for generating the training data for Machine Learning Algorithms: capturing raw data from the network interfaces, classification by port numbers, Deep Packet Inspection, and statistical classification. It was shown that the current methods are not able to provide high-quality training data. Capturing raw data from network interfaces is not scalable, since it requires installing and testing one application at a time; the background traffic (as DNS requests, system upgrades) also is a concern, as the resulting dataset is highly polluted. Classification by ports is not able to identify applications, which use dynamic port numbers. It also cannot distinguish between different applications using the same port number and it is easy to cheat by moving an application to another port. Deep Packet Inspection is slow and it requires a lot of processing power, unable to inspect encrypted or tunneled traffic, and it causes a lot of privacy and legal concerns. Therefore, for the collection of the training data, we used our Volunteer-Based System (VBS).

The real-time classification approach is as follows. At first, we recruit some volunteers from the users of the particular network. We assume that the users within the same network use the network in a similar way, for example, they use similar applications. The volunteers install VBS clients on their computers. The per-application statistics are used as training data for C5.0 MLA, which is able to distinguish these applications in the core of the network, while inspecting the traffic from all the network users. Then, knowing the application associated with a particular flow, we can assess the quality of the application connection. The paper presents the design of the centralized monitoring solution in the network core, which makes use of both passive and active measurements techniques to assess various QoS parameters.

Paper IV – Classification of HTTP Traffic Based on C5.0 Machine Learning Algorithm

As mentioned in Paper I, VBS was extended of the possibility to split the HTTP flows into logical parts, which transport different files or carry different streams. Each file or stream transported by HTTP is associated with its content type using the information from the *content-type* field contained by the HTTP header. In Paper II, we performed experiments with C5.0 classification of 7 groups of traffic. One of the groups was a browser traffic in general, selected based on the application name (as *chrome*, *firefox*, or *iexplore*). However, it was emphasized that the browser traffic is diverse: it can contain data resulted from interactive web browsing, file downloads, radio or video streaming, embedded FTP or BitTorrent clients, etc. Therefore, in this paper, we tried to concentrate on approaches to distinguish the different kinds of the browser traffic in an automatic way based on the information extracted from HTTP headers by VBS.

The identification of the browser traffic was intended to be done based on the type of the transferred content and the character of the traffic. Therefore, based on the *content-type* field contained by the HTTP header and the character of the traffic, we specified 5 groups of browser traffic: *audio*, *file download*, *multimedia*, *video*, and *web*. We need to emphasize that one element could fall into two groups, as for example, a downloaded video file. In this case, we assigned the element to the group based on the type of the content, as we were not able to see if an element marked as *audio/mpeg* is in fact an audio stream from a web radio, or a downloaded MP3 file. Apart from that, we extended the traffic groups by including traffic originated by other applications than web browsers: the *video* class was enriched by adding the traffic from Flash streaming browser plugins, the *audio* class was enriched by adding Skype, and the *file download* class was enriched by adding several applications, which are used to transfer files. We also added 2 new groups: *ssh* and *p2p*. The second group also contained some elements, which could be identified as *file download*, but we were not able to pre-select them from the *p2p* traffic. The classification experiment was made in almost the same manner and using the same classification attributes as in Paper II. The only difference concerns the way of obtaining the chunks of 35 packets, which are used to create the training and test cases for the classifier. Non-HTTP flows are treated exactly in the same way as in Paper II. However, HTTP flows are split into parts corresponding to the individual files or streams, and each of the elements is treated separately according to its content. Therefore, we were able to take samples of the video files, audio files, other binary files, etc. However, it does not make sense to split the web browsing activity into particular HTML, JPG, CSS, and other files, as all of them are rather short and they are a part of the same traffic patterns. Therefore, the web browsing flows were not separated, but treated as a whole (*web* class) in the same manner as non-HTTP flows.

The classification results are as follows. Taking into account only the browser traffic, we obtained 17 % of errors, while taking into account all the traffic, the error rate was

6 %. Almost all the cases from the *audio*, *web*, *p2p*, and *ssh* groups were classified properly. However, we obtained a big number of misclassifications between the *file download* and *video* groups for the browser traffic. This can be explained logically: most of the video traffic transmitted by HTTP are just regular transfers of video files, as YouTube does – in this case the video file is at first downloaded (entirely or in part) to the users' computers and then replayed by the web browsers or their plugins. Therefore, the behavior of such video transfers (as from YouTube) is identical to the transfers of other files, for example, executables or compressed ZIP archives. However, video streamed by RTMP protocol by external browser plugins can be easily separated from the *file download* traffic, as its behavior is totally different.

We also showed statistical profiles of flows transmitting different types of HTTP contents. We showed that the profiles do not depend on the type of the transmitted content, but on the particular application or web service. We demonstrated that there is no difference between the profiles of transfers of video files (f.x. from YouTube) and transfer of other big binary files (as ZIPs) by HTTP and, therefore, different content types cannot be distinguished based on the statistical attributes. However, the paper shows that we can identify the content if it is associated with a particular protocol or behavior, as for example we can recognize video streaming (by RTMP protocol) or audio streaming (by both RTMP and HTTP).

Paper V – Obtaining Internet Flow Statistics by Volunteer-Based System

This paper focuses on generating various statistics of the network traffic on the flow level. Such statistics can be used for modeling of the traffic or the particular applications, creating realistic scenarios of future Internet usage, or analysis of the impact of the particular traffic type on the performance of the network. To obtain the statistical data, 4 users (2 in Denmark and 2 in Poland) were monitored during 5 months by VBS. At first, for each user (and for all users altogether), we calculated the distribution of the flows among TCP and UDP, and the average lengths and durations of TCP and UDP flows. Afterwards, we selected the top 5 most popular applications for each user separately and for all users altogether, and we presented the number of flows generated by these applications as the number and the percentage of the total amount of flows from the particular user. The most popular application was uTorrent, which amounted for 74.12 % of all flows from all user altogether. It was also the most popular application for 3 out of 4 examined users, where its share among of the generated flows was respectively 89.43 %, 91.36 %, and 62.78 %. At the end, the cumulative distribution of flows for all users over time was presented.

Paper VI – Obtaining Application-Based and Content-Based Internet Traffic Statistics

This paper is a continuation of the work presented in Paper V. However, instead of dealing with the statistics on the flow level, we emphasize the possibility of obtaining statistics of various applications and types of files transferred by HTTP regarding both the number of flows and the volume of traffic. For our experiments, we used the same traffic from the same users as in Paper V. At first, we compared the number of flows versus the traffic volume from the particular users over time. Generally, when the number of flows increases, the data volume increases as well. However, some users are characterized by bigger number of light flows, while some users are characterized by a smaller number, but longer and heavier flows.

Then, we presented 10 top applications for all users altogether and for each user separately. For each application, we calculated the number of flows and the data volume as well as the percent values of the total number of flows / data volume from the particular user. We also calculated the average number of packets in a flow created by the applications. There are several interesting observations regarding the most used applications and the data collected by VBS. The average number of packets in flows without assigned application name is 11, comparing to 63–139 567 in flows, where the application name is assigned – the data in flows without the assigned application name amounts for 11 % of flows, but only 1 % of the traffic volume. Moreover, applications responsible for streaming video are shown to generate small number of flows, but containing a big number of packets. In our case, the proportion of the number of flows generated by video streaming applications to the group without the associated application name was 1:12 898, while the proportion of the data volume was 2.75:1. We looked carefully at the behavior of uTorrent application, which was responsible for generating the biggest number of flows (72 % of total) and the biggest data volume (61 % of total). The amount of downloaded data was around 7 times bigger than the amount of uploaded data, while the upload and download was conducted at the same time. By looking at the download to upload rate, we can deduce if the user uses symmetric or asymmetric Internet connection.

The same experiments were repeated for the most popular *content-types* of files transported by HTTP. The results showed that the majority of HTTP traffic was generated by video files and binary files downloaded by the users. The web browsing traffic (*image/jpeg*, *text/html*, and *text/plain*) accounted for 52 % of the number of files transported by HTTP, but only for 9 % of the data volume, primarily because they contain a low number of packets (4–9 in average). Finally, the volume of the traffic generated by *video/x-flv* files was around 2.5 times bigger than the volume of the traffic generated by the second biggest contributor.

We also made a summary of some statistical parameters for different kinds of applications. We calculated average packet sizes (inbound, outbound, total), distribution of inbound and outbound packets, and distribution of inbound and outbound packets

carrying data.

Paper VII – Is our Ground-Truth for Traffic Classification Reliable?

Classification of computer network traffic rapidly increased its significance during the last years, becoming a key aspect of many network related tasks. Therefore, many different classification methods and tools were developed or are under development. Testing of the accuracy of the classifiers become a challenging issue, as it requires possession of clean data samples from various applications. The paper begins by describing the current methods for obtaining the ground-truth and it assesses their advantages and drawbacks. It is said that the most common technique is to create a proprietary dataset by labeling the traffic by Deep Packet Inspection tools, or to use an already available dataset, which is usually pre-classified by DPI as well. This approach, however, creates a kind of a loop: DPI tools are used to create a dataset, which is used to test traffic classifiers, including the DPI tools themselves.

This paper has two major contributions. At first, by using VBS, we created a dataset of 10 different applications (eDonkey, BitTorrent, FTP, DNS, NTP, RDP, NETBIOS, SSH, HTTP, RTMP), which is available to the research community. It contains 1 262 022 flows captured during 66 days. The HTTP flows are captured from several different web services. The dataset is available as a bunch of PCAP files containing full flows including the packet payload, together with corresponding text files, which describe the flows in the order as they were originally captured and stored in the PCAP files. The description files contain the start and end timestamps of flows based on the opening and closing of the system sockets, which is useful to reproduce the original behavior, when many short flows are generated between the same hosts during a short period of time. Furthermore, the application name taken from the system sockets is appended, which sets the ground-truth label of the flows. This dataset can be directly used to test various traffic classifiers: port-based, DPI, statistical, etc.

At second, we examined the ability of selected Deep Packet Inspection tools to accurately label network flows in order to create a dataset composed of selected applications, which can be used as a ground truth for testing various traffic classifiers. The tested DPI tools are: PACE, OpenDPI, L7-filter, nDPI, Libprotoident, and NBAR. The test was performed on our dataset labeled by VBS, which eliminated the possibility of having mistakes in the reference data. As the test was made only to assess the ability of the DPIs to perform an accurate classification to create a ground-truth dataset, we did not test other properties, as the resource usage, or time used to classify the dataset. All the classifiers except NBAR were tested by a special benchmark tool, which read the PCAP files together with their descriptions, composed the packets in the original flows, and provided the flows to the DPIs organized as libraries. To test the accuracy of NBAR, we needed to emulate a Cisco router by using Dynamips together with an original Cisco

Internetwork Operating System image. The packets needed to be replayed back to the virtual interface where the Cisco router resided in order to be classified by NBAR. That imposed a few new requirements. At first, the destination MAC address of each packet needed to be changed to the MAC address of the virtual Cisco router interface, as Cisco routers do not accept packets, which are not directed to their interfaces. At second, the source MAC addresses were changed to contain the identifiers of the original flows, so the router could re-construct and assess the flows as they were originated. Then, the Flexible NetFlow feature of Cisco routers was used to apply per-flow application label by NBAR. The NetFlow records were captured on the host machine, where they were analyzed.

The obtained results present *PACE*, a commercial tool, as the most accurate solution for ground-truth generation (91.07 % of average accuracy on our dataset). However, among the open-source tools available, *Libprotoident* (84.16 %), and *nDPI* (82.48 %) also achieve very high accuracy. On the other hand, *L7-filter* (38.13 %) and *NBAR* (46.72 %) should not be used to obtain the ground-truth.

Paper VIII – Multilevel Classification and Accounting of Traffic in Computer Networks

Classification and accounting of computer network traffic is an important task of Internet Service Providers, as it allows for adjusting the bandwidth, the network policies, and providing better experience to their customers. However, existing tools for traffic classification are incapable of identifying the traffic in a consistent manner. The results are usually given on various levels for different flows. For some of them only the application is identified (as *HTTP*, *BitTorrent*, or *Skype*), for others only the content (as *audio*, *video*) or content container (as *Flash*), for yet others only the service provider (as *Facebook*, *YouTube*, or *Google*). Furthermore, Deep Packet Inspection (DPI), which seems to be the most accurate technique, in addition to the extensive needs for resources, often cannot be used by ISPs in their networks due to privacy or legal reasons. Techniques based on Machine Learning Algorithms (MLAs) require good quality training data, which are difficult to obtain. MLAs usually cannot properly deal with other types of traffic, than they are trained to work with – such traffic is identified as the most probable class, instead of being left unclassified. Another drawback of MLAs is their inability to detect the content carried by the flow, or the service provider.

To overcome the drawbacks of already existing methods, we developed a novel hybrid method to provide accurate identification of computer network traffic on six levels: *Ethernet*, *IP protocol*, *application*, *behavior*, *content*, and *service provider*. The *Ethernet* and *IP protocol* levels are identified directly based on the corresponding fields from the headers (*EtherType* in Ethernet frames and *Type* in IP packet). The *application* and *behavior* levels are assessed by a statistical classifier based on C5.0 Machine Learning Algorithm. Finally, *content* and *service provider* levels are identified based on IP ad-

dresses. The training data for the statistical classifier and the mappings between the different types of content and the IP addresses are created based on the data collected by Volunteer-Based System, while the mappings between the different service providers and the IP addresses are created based on the captured DNS replies. Support for the following applications is built into the system: America's Army, BitTorrent, DHCP, DNS, various file downloaders, eDonkey, FTP, HTTP, HTTPS, NETBIOS, NTP, RDP, RTMP, Skype, SSH, and Telnet. Within each application group we identify a number of behaviors – for example, for HTTP, we selected *file transfer*, *web browsing*, *web radio*, and *unknown*. Our system built based on the method provides also traffic accounting and it was tested on 2 datasets.

The classification results are as follows. On the *Ethernet* and *IP protocol* levels we achieved 0.00 % errors. The classification on the *application* and *behavior* levels were assessed together. Using the first dataset, we achieved 0.08 % of errors, while 0.54 % of flows remained as unknown. Using the second dataset, we achieved 0.09 % of errors, while 0.75 % of flows remained as unknown. Taking into account the *content* level, the classification using the first dataset gave us 0.22 % errors and 0.47 % of unclassified flows, while using the second dataset it gave us 0.96 % of errors and 1.42 % of unclassified flows. The classification on the *service provider* level was performed only using the first dataset (we needed the application-layer payloads) and it gave us 1.34 % of errors and 1.71 % of unknown flows. Therefore, we have shown that our system gives a consistent, accurate output on all the levels. We also showed that the results provided by our system on the application level outperformed the results obtained from the most commonly used DPI tools. Finally, our system was implemented in Java and released as an open-source project.

Paper IX – Independent Comparison of Popular DPI Tools for Traffic Classification

Network traffic classification has become an essential input for many network-related tasks. However, the continuous evolution of the Internet applications and their techniques to avoid being detected (as dynamic port numbers, encryption, or protocol obfuscation) have considerably complicate their classification. We start the paper by introducing and shortly describing several well-known DPI tools, which later will be evaluated: *PACE*, *OpenDPI*, *L7-filter*, *nDPI*, *Libprotoident*, and *NBAR*. We tried to use the most recent versions of the classifiers. However, OpenDPI project was closed in June 2011 and since that time no new version of this software was released. L7-filter, which is broadly described in the scientific literature, also seems to be not developed any longer – the most recent version of the classification engine is from January 2011 and the classification rules from 2009.

This paper has several major contributions. At first, by using VBS, we created 3 datasets of 17 application protocols, 19 applications (also various configurations of the

same application), and 34 web services, which are available to the research community. The first dataset contains full flows with entire packets, the second dataset contains truncated packets (the Ethernet frames were overwritten by 0s after the 70th byte), and the third dataset contains truncated flows (we took only 10 first packets for each flow). The datasets contain 767 690 flows labeled on a multidimensional level. The included application protocols are: DNS, HTTP, ICMP, IMAP (STARTTLS and TLS), NETBIOS (name service and session service), SAMBA, NTP, POP3 (plain and TLS), RTMP, SMTP (plain and TLS), SOCKSv5, SSH, and Webdav. The included applications (and their configurations) are: 4Shared, America's Army, BitTorrent clients (using plain and encrypted BitTorrent protocol), Dropbox, eDonkey clients (using plain and obfuscated eDonkey protocol), Freenet, FTP clients (in active and passive modes), iTunes, League of Legends, Pando Media Booster, PPLive, PPStream, RDP clients, Skype (including audio conversations, file transfers, video conversations), Sopcast, Spotify, Steam, TOR, and World of Warcraft. The included web services are: 4Shared, Amazon, Apple, Ask, Bing, Blogspot, CNN, Craigslist, Cyworld, Doubleclick, eBay, Facebook, Go.com, Google, Instagram, Justin.tv, LinkedIn, Mediafire, MSN, Myspace, Pinterest, Putlocker, QQ.com, Taobao, The Huffington Post, Tumblr, Twitter, Vimeo, VK.com, Wikipedia, Windows Live, Wordpress, Yahoo, and YouTube.

These datasets are available as a bunch of PCAP files containing full flows including the packet payload, together with corresponding text files, which describe the flows in the order as they were originally captured and stored in the PCAP files. The description files contain the start and end timestamps of flows based on the opening and closing of the system sockets, which is useful to reproduce the original behavior, when many short flows are generated between the same hosts during a short period of time. The application name taken from the system sockets is appended as well. Furthermore, each flow is described by one or more labels defining the application protocol, application itself, or the web service. These datasets can be directly used to test various traffic classifiers: port-based, DPI, statistical, etc.

At second, we developed a method for labeling non-HTTP flows, which belong to web services (as YouTube). In our previous approach shown in Paper VII, the labeling was done based on the corresponding domain names taken from the HTTP header. However, that could allow to identify only the HTTP flows. Other flows (as encrypted SSL / HTTPS flows, RTMP flows) were left unlabeled. Therefore, we implemented a heuristic method for detection of non-HTTP flows, which belong to the specific services. To be recognized as a non-HTTP web flow, the application name associated with the flow should be the name of the web browser (as *chrome*), a name of a web browser plugin (as *plugin-container*, *flashgcplay*), or the name should be missing. Then, we looked at the HTTP flows, which were originated from 2 minutes before to 2 minutes after the non-HTTP web flow. If all the corresponding (originated from the same local machine and reaching the same remote host) HTTP flows had a web service label assigned, and the service label was the same for all of the flows, the non-HTTP flow obtained the

same web service label. That allows us to test how the classifiers identify encrypted and other non-HTTP traffic belonging to various web services.

Then, we examined the ability of the DPI tools to accurately label the flows included in our datasets. All the classifiers except NBAR were tested by a special benchmark tool, which read the PCAP files together with their descriptions, composed the packets in the original flows, and provided the flows to the DPIs organized as libraries. To test the accuracy of NBAR, we needed to emulate a Cisco router by using Dynamips together with an original Cisco Internetwork Operating System image. The packets needed to be replayed back to the virtual interface where the Cisco router resided in order to be classified by NBAR. That imposed a few new requirements. At first, the destination MAC address of each packet needed to be changed to the MAC address of the virtual Cisco router interface, as Cisco routers do not accept packets, which are not directed to their interfaces. At second, the source MAC addresses were changed to contain the identifiers of the original flows, so the router could re-construct and assess the flows as they were originated. Then, the Flexible NetFlow feature of Cisco routers was used to apply per-flow application label by NBAR. The NetFlow records were captured on the host machine, where they were analyzed.

It was shown that the detection rate is almost identical on the set containing full flows with entire packets and the set with truncated flows, while it highly decreases on the set with truncated packets. However, Libprotoident is an exception, as it provides the same results independent of the set, as it uses only 4 B of packet payload. We showed that, in most cases NBAR (apart of Libprotoident) was the most resistant tool regarding the impact of packet truncation on the detection rate.

We showed that PACE is able to identify the highest number of various web services among all the studied classifiers. PACE detected 16 web services, nDPI 10, OpenDPI 2, L7-filter in its default version only 1, Libprotoident 1, and NBAR none. We have also shown that L7-filter is characterized by a very high number of misclassified flows belonging to web services (usually 80–99 %) – the flows were recognized in a vast majority as *Finger* and *Skype*.

We evaluated the impact of protocol encryption or obfuscation on the detection rate by the particular classifiers. Protocol encryption made the detection rate lower in all the cases, while we did not see such dependency while using obfuscated eDonkey protocol – in this case, PACE demonstrated even increased detection rate from 16.50 % (for plain traffic) to 36 %. We have shown that only PACE is able to identify accurately some applications, which are supposed to be hard to detect, as Freenet or TOR.

Paper X – nDPI: Open-Source High-Speed Deep Packet Inspection

This paper begins by introducing various methods for passive traffic identification. Apart from the obvious drawbacks of port-based classification methods, the paper em-

phasizes two major drawbacks of Machine Learning based tools. At first, they are able to classify only a few traffic categories, being less suitable for fine protocol granularity detection applications. At second, their inaccuracy makes them less useful in mission critical applications, as traffic blocking or precise accounting. For those reasons, there is a need for a lightweight, fast, and accurate Deep Packet Inspection tool.

This paper introduces nDPI – an open source library written in C, built based on the OpenDPI GPL-licensed code. The differences between OpenDPI and nDPI are described, focusing on the improvements and new features added to nDPI. The current design and implementation of the classifier is described in detail, including the way how the encrypted traffic is handled. Namely, the certificates associated with SSL connections are decoded in order to extract the host name. Additionally, some encrypted connections are detected based on the IP addresses.

In the last versions of nDPI (svn revision 7249 or newer), we decided to remove the use of Skype and BitTorrent heuristics, so that we eliminated false positives at the cost of slightly increasing the number of undetected flows when using these two protocols. We also re-implemented support for several other protocols, as FTP, SOCKSv4, SOCKSv5, eDonkey, PPLive, Steam, RTMP, and Pando. Furthermore, we added the ability to discover new web services, e.g., Wikipedia and Amazon. Finally, the classifier was validated against 31 popular protocols and applications, and 7 commonly used web services. We showed that the classifier in the current version is characterized by high accuracy (in half of the cases approaching 100 %) and a very low rate of misclassified flows (for most classes less than 1 %). We demonstrated that the test application using nDPI processes packets at an average speed of 3.5 Mpps / 8.85 Gbps using a single core CPU and a PCAP file as the packet source. In terms of memory usage, nDPI needs some memory to load the configuration and automatas used for string-based matching. This memory used by nDPI is ~210 KB with no custom configuration loaded that increases of ~25 KB when a custom configuration is loaded. In addition to that, nDPI keeps per-flow information that is independent from the application protocols and which takes ~1 KB per flow.

As other DPI tools, nDPI also has some drawbacks. Habitually, the classifiers provide only result per flow, which is supposed to characterize the flow in the most detailed manner. Therefore, the output is a mix of results on various levels: IP protocols (i.e., TCP or UDP), application protocols (e.g., DNS, HTTP, SSL, BitTorrent, or SMTPS), types of the content (e.g., MPEG, or Flash), and finally, the service providers (e.g., Facebook, or YouTube). The usefulness of such result is very limited. At first, one flow can use several application protocols (as HTTP and Dropbox). At second, one application protocol (as DNS) can use several IP protocols (TCP and UDP). At third, it is impossible to judge if the most detailed level is the content (as Flash) or the service (as YouTube). Finally, this scheme does not allow to provide precise accounting of the traffic (for example, it is not possible to account the HTTP traffic, if the results are given on multiple levels, so in majority of cases HTTP is even not mentioned). At the

end of the paper, we proposed to make the nDPI classifier consistent. Many consistency fixes were already applied, which resulted in creating nDPIng. More about this project can be read in Section 7.1.

6 Conclusion

This thesis covers several broad areas in traffic monitoring and analysis both on the user side and in the network. We designed, developed, and evaluated a host-based traffic monitoring tool, called Volunteer-Based System (VBS), which provides detailed statistics about the traffic passing through the network connections. That gives an overview how the network is utilized, allows to better allocate the available bandwidth, tune the network parameters on routers and switches, or to create special facilities for particular application or services (as proxy or cache servers, local DNS servers). The data collected by VBS can be used to create realistic traffic profiles of the selected applications. They can be used to build application traffic generators, where the characteristics of the generated traffic match the real characteristics of the simulated applications. They also can be the ground-truth for testing and building new network traffic classifiers. A special version of VBS, which collects full Ethernet frames, can be also used to evaluate the accuracy of Deep Packet Inspection classification tools.

We assessed the usefulness of C5.0 Machine Learning Algorithm (MLA) in the classification of computer network traffic. MLAs require good quality training data in order to be able to create accurate classification rules. Therefore, as the source of the training data, we used the information collected by VBS. We showed that the application-layer payload is not needed to train the C5.0 classifier to be able to distinguish different applications in an accurate way. Statistics based on the information accessible in the headers and the packet sizes are fully sufficient to obtain high accuracy. Our method was shown to reach 99.90 % accuracy while recognizing the traffic belonging to 7 different groups. We tested various classification modes (decision trees, rulesets, boosting, softening thresholds) regarding the classification accuracy and the time required to create the classifier. We also assessed the dependency between the number of training cases (and the number of packets in the case) and the classification accuracy by C5.0. We showed how the information from the *content-type* field in HTTP headers can be used to split the transport-layer flows into parts transporting different files.

We demonstrated how to obtain per-flow, per-application, and per-content statistics of traffic in computer networks. Our VBS was used to obtain overall and per-user statistics based on flows. We showed the distribution, average lengths, and durations of TCP and UDP flows, and the distribution of flows belonging to top 5 applications. Finally, the cumulative number of flows for each user over the time was shown on the graph to provide the overview of the users' network activity during several months. We also included statistics regarding the traffic volume. It enabled us to compare the number of flows and their distribution with the corresponding amount of data for

all monitored users altogether and for each user separately. We also included in the evaluation the statistics regarding transferred files by HTTP. Therefore, we identified the most common types of files transferred by HTTP and presented which share of the overall HTTP traffic they represent.

We created two datasets composed of different applications, which can be used to assess the accuracy of different traffic classification tools. The datasets contain full packet payloads and they are available to the research community as a set of PCAP files and their per-flow description in the corresponding text files. The included flows were labeled by VBS. We developed a method for labeling non-HTTP flows, which belong to web services (as YouTube). Classically, labeling was done based on the corresponding domain names taken from the HTTP header. However, that could allow to identify only the HTTP flows. Other flows (as encrypted SSL / HTTPS flows, RTMP flows) were left unlabeled. Labeling of these flows allowed us to test how different classifiers identify encrypted and other non-HTTP traffic belonging to various web services.

We evaluated the ability of several Deep Packet Inspection tools (PACE, OpenDPI, L7-filter, nDPI, Libprotoident, and NBAR) to label flows in order to create datasets serving as a ground-truth for the subsequent evaluation of various traffic classification tools. We showed that PACE achieves the best classification accuracy from all the tested tools, while the best performing open-source classifiers are nDPI and Libprotoident. L7-filter and NBAR, however, should not be considered as reliable tools. We also described the methodology of testing Cisco NBAR, which is difficult to evaluate, because it works only on Cisco devices. Therefore, we presented how to re-play the collected packets from PCAP files back to the network in a way that the router is able to inspect them. We also described how to configure the Cisco router to inspect the traffic by NBAR with Flexible NetFlow, which is able to generate per-flow records. Finally, we showed how to capture NetFlow records from the router, configure the software for inspecting the NetFlow records, and generate a readable output.

We designed, developed, and evaluated a multilevel traffic classifier. The classification is performed on six levels: *Ethernet*, *IP protocol*, *application*, *behavior*, *content*, and *service provider*. The *Ethernet* and *IP protocol* levels are identified directly based on the corresponding fields from the headers (*EtherType* in Ethernet frames and *Type* in IP packet). The *application* and *behavior* levels are assessed by a statistical classifier based on C5.0 Machine Learning Algorithm. Finally, the *content* and *service provider* levels are identified based on IP addresses. The system is able to deal with unknown traffic, leaving it unclassified on all the levels, instead of assigning the traffic to the most fitting class. The training data for the statistical classifier and the mappings between the different types of content and the IP addresses are created based on the data collected by VBS, while the mappings between the different service providers and the IP addresses are created based on the captured DNS replies. Our system was implemented in Java and released as an open-source project. The accuracy of the classification on the application level by our system was compared with the accuracy given by several

DPI tools on 2 datasets and we demonstrated that our classifier outperforms the DPI tools regarding the applications, which it supports.

We directly compared the ability of several DPI tools (PACE, OpenDPI, two configurations of L7-filter, nDPI, Libprotoident, and NBAR) to classify 17 application protocols, 19 applications (also various configurations of the same application), and 34 web services on a dataset of 767 690 labeled flows. We evaluated the impact of flow or packet truncation on the detection rate by the particular classifiers. We tested the classifiers on 3 sets of data: containing full flows with entire packets, with truncated packets (the Ethernet frames were overwritten by 0s after the 70th byte), and with truncated flows (we took only 10 first packets for each flow). We evaluated the impact of protocol encryption or obfuscation on the detection rate by the particular classifiers. We have also shown that only PACE is able to identify accurately some applications, which are supposed to be hard to detect, as Freenet or TOR.

We also introduced an alternative strategy of traffic identification. nDPI, a high-performance traffic classification tool based on Deep Packet Inspection (DPI), was demonstrated to be able to process packets at an average speed of 3.5 Mpps / 8.85 Gbps using a single core CPU and a PCAP file as the packet source. After implementing support for new applications, the classifier was validated against 31 popular protocols and applications, and 7 commonly used web services. We showed that the classifier in the current version is characterized by high accuracy (in half of the cases approaching 100 %) and a very low rate of misclassified flows (for most classes less than 1 %). We also proposed future enhancements, which make the nDPI classifier consistent regarding the generated results.

Our last focus was on designing a method for assessing the Quality of Service in computer networks. Our method relies on VBS clients installed on a representative group of users from the particular network. The per-application traffic profiles obtained from the machines belonging to the volunteers are used to train the C5.0 Machine Learning based tool to recognize the selected applications in any point of the network. After the application is being identified, the quality of the application session can be assessed. For that purpose, we proposed a hybrid method based on both passive and active approaches.

Bredbånd Nord, a regional fiber networks provider, was our collaboration partner throughout the PhD study. For us, the collaboration benefits included the access to real network data and real production environment, which resulted in the development of an accurate traffic monitoring solution. The first part of our collaboration was focused on the development of VBS. The network users provided by the company participated in testing the software and volunteered by giving us access to their network data. The second part of our collaboration was focused on the development of a traffic accounting solution, which would be able to provide an overview how the network is utilized. We contributed by designing and development of the multilevel traffic classifier. Bredbånd Nord contributed by giving us access to anonymized one-week traces of the DNS traffic

in their network. Thanks to that, we were able to obtain the mappings between the queried domain names and the IP addresses as well as the number of queries for the particular domain.

7 Future Work

Our methods and the developed by us solutions are a field for constant improvements.

The first possibility could be to enhance our Volunteer-Based System (VBS). Sufficient security mechanisms should be implemented, so that the system would stop being prone to SQL injection. Mutual authentication mechanisms between the clients and the server should be designed in order to minimize the chances of pushing an unauthorized upgrade to the clients or sending a poisoned data file to the server. If it is possible, we should avoid using external tools (Netstat, TCPView), but obtain the information about the open sockets directly from the system API. That would remove the limitation of the minimum flow duration of 1 second in order to allow the flow being noticed by VBS. We need to consider developing an intelligent transfer protocol, which will take care of negotiating link parameters and of scheduling transfers in order to effectively use the capacity of the link. An automatic Linux installer will be the next step to make the system more friendly to non-qualified users. For now, VBS requires a valid IPv4 address to listen on a network interface, but IPv6 is also planned to be supported. Another issue arises when an encapsulation, data tunneling or network file systems (like SAMBA, NFS) are used. Then, only the most outer IP and TCP/UDP headers are inspected.

Another important part is the recruitment of more volunteers, in order to collect larger amounts of data. Also, having appropriate background information about the users could be useful. This includes both the data about the users themselves, such as age, occupation, if the computer is shared, but also information about the connection, as speeds and technologies.

The data collected by VBS was used to generate a number of statistics on the flow and data volume level. These statistics were calculated per-flow, per-application, and per-content. Future research will focus on developing efficient methods for extracting relevant information from the packet statistics. This can provide even more valuable information about the flows, for example, on average packet sizes of different flows (and the distribution of packet sizes), inter-arrival times between packets, and the number of successful vs. unsuccessful connections for different kinds of traffic. Moreover, particularly interesting statistics can be derived from the combined flow and packet statistics, such as the average size of flows of different kinds of traffic, and eventually how much traffic is created by different applications for individual users. The applications and the files of various content-types transmitted by HTTP can also be grouped into several sets, like voice, video, file transfer, interactive browsing, etc. This is not a trivial task, since grouping manually such large number of applications and content-types is not doable. The challenge is that it is large amounts of data, so efficient ways of handling these has

to be developed.

A significant part of our research was focused on the comparison of various Deep Packet Inspection (DPI) tools. Although this study is complete, the continuous evolution of the network applications and the DPI-based techniques allows a periodical updated of the evaluation. For instance, this evaluation can be updated by adding new applications and web services to the dataset (as Netflix) and by introducing new classification tools to the study (as NBAR2 or Tstat). In this thesis, we focused on the reliability of the DPI tools, however, a possible line of future work can be related to their deployment for real-time classification (as scalability and computational cost).

Our hybrid network classifier was shown to be able to identify the traffic on multiple levels with high accuracy. However, the number of supported applications is very limited. Therefore, the next step would be to collect the traffic from other applications and include them in our system. An automatic retraining mechanism can be developed to update the mappings between the domain names and the web services.

The next big potential field for further research and development is our method for QoS assessment. For now, only the traffic classification part is fully implemented. However, we did not perform any studies regarding the impact of estimators (as delay, jitter, burstiness, or packet loss) on the performance of any particular application. Furthermore, as some of the estimators (as delay and packet loss) can only be measured actively for non-TCP traffic, the future research can be directed into that area.

7.1 nDPIng

nDPIng project was started as a separate branch of nDPI to address these weaknesses that require major changes in the code. As the original nDPI, nDPIng is GPLv3-licensed and accessible at https://svn.ntop.org/svn/ntop/trunk/nDPIng/. The project is now in an early development alpha stage, so not all the protocols and applications detected by nDPI are supported. As the current nDPIng API is different than the one of nDPI (and still provisional), the existing applications designed to work with nDPI will not work with nDPIng without changing the way how the library is used.

The aim of this unique project is to bring new quality to the field of traffic classification by providing consistent results on many levels. The clear, unambiguous identification of network flows is meant to be ensured by various classification techniques combined into a single tool. The following information is intended to be given for each flow inspected by the classifier:

1. Transport layer protocol
2. All the application-layer protocols
3. Type of the content
4. Service provider

5. Content provider

To demonstrate the abilities of the software, we show a few examples of the currently obtained results. The current format of the results is not the final one, however, it demonstrates various new techniques used by nDPIng:

- `proto:TCP→SSL_with_certificate→POP3S, service:Google`
  An encrypted POP3 session with a Google mail server.
- `proto:TCP→SSL_with_certificate, service:Twitter`
  An encrypted connection to a Twitter server.
- `proto:TCP→FTP_Data, content:JPG`
  An FTP data session, which carries a JPG image.
- `proto:TCP→SSL_with_certificate→Dropbox, service:Dropbox`
  An encrypted Dropbox session (the application is Dropbox) with a Dropbox server.
- `proto:TCP→SSL_with_certificate, service:Dropbox`
  An encrypted session with a Dropbox server, while the application is unknown (it can be Dropbox as well as a web browser).
- `proto:TCP→HTTP, content:WebM, service:YouTube`
  A flow from YouTube, which transports a WebM movie.

7.2 Quality of Service in Wireless Local Area Networks

Another important application of traffic classification are Wireless Local Are Networks (WLANs). The 802.11e standard defines four traffic access classes: voice, video, best-effort, and background [41]. The QoS is accomplished by creating inside a wireless node separate queues for each of these classes. The queues behave like individual wireless nodes, competing for the access to the medium according to the Enhanced Distributed Channel Access (EDCA) algorithm. It ensures that the high-priority traffic is more likely to be sent than the low-priority traffic by applying different values of parameters (as contention window, arbitration inter-frame spacing, or transmission opportunity) to the particular queues [42]. The flows are mapped to the access classes directly based on the Class of Service (CoS) field from the Ethernet frame [43] or Type of Service (ToS) field from the IP packet [44], depending on the device.

Based on that, we can enumerate several issues, which are subject to further investigation. As the appropriate access class is chosen based on the values from the Ethernet or IP headers, the accuracy of the association highly depends on the quality of pre-classification and policies inside the network. Wireless nodes directly connected to the users' devices are required to rely on the QoS tagging done by the users' applications or the operating system. We observed a significant amount of BitTorrent traffic

coming from the users tagged as Expedited Forwarding (EF), and thus, falling into the *voice* access class. As shown in Paper IV, the QoS rules should be applied based on the character of the flow and the content, not on the content alone. While it is worth to prioritize live video RTMP streams, there is no need to do that with video files downloaded or uploaded using HTTP, FTP, or BitTorrent. Therefore, we can question the validity of the content-based mappings.

References

[1] Kartheepan Balachandran, Jacob Honoré Broberg, Kasper Revsbech, and Jens Myrup Pedersen. Volunteer-Based Distributed Traffic Data Collection System. In *Proceedings of the 12th International Conference on Advanced Communication Technology (ICACT 2010)*, volume 2, pages 1147–1152. IEEE, Phoenix Park, PyeongChang, Korea, February 2010.

[2] Kartheepan Balachandran and Jacob Honoré Broberg. Volunteer-Based Distributed Traffic Data Collection System. Master's thesis, Aalborg University, Department of Electronic Systems, Denmark, June 2010.

[3] Riyad Alshammari and A. Nur Zincir-Heywood. Machine Learning based encrypted traffic classification: identifying SSH and Skype. In *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA 2009)*, pages 1–8. IEEE, Ottawa, Ontario, Canada, July 2009. DOI: 10.1109/CISDA.2009.5356534.

[4] Sven Ubik and Petr Žejdl. Evaluating application-layer classification using a Machine Learning technique over different high speed networks. In *Proceedings of the Fifth International Conference on Systems and Networks Communications (ICSNC)*, pages 387–391. IEEE, Nice, France, August 2010. DOI: 10.1109/ICSNC.2010.66.

[5] Jun Li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. Real-time P2P traffic identification. In *Proceedings of the IEEE Global Telecommunications Conference (IEEE GLOBECOM 2008)*, pages 1–5. IEEE, New Orleans, Louisiana, USA, December 2008. DOI: 10.1109/GLOCOM.2008.ECP.475.

[6] Ying Zhang, Hongbo Wang, and Shiduan Cheng. A Method for Real-Time Peer-to-Peer Traffic Classification Based on C4.5. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 1192–1195. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689126.

[7] Jing Cai, Zhibin Zhang, and Xinbo Song. An analysis of UDP traffic classification. In *Proceedings of the 12th IEEE International Conference on Communication*

Technology (ICCT), pages 116–119. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689203.

[8] Riyad Alshammari and A. Nur Zincir-Heywood. Unveiling Skype encrypted tunnels using GP. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, Barcelona, Spain, July 2010. DOI: 10.1109/CEC.2010.5586288.

[9] L7-filter Supported Protocols, 2012. [Online]. Available: `http://l7-filter.sourceforge.net/protocols`.

[10] Li Jun, Zhang Shunyi, Lu Yanqing, and Zhang Zailong. Internet Traffic Classification Using Machine Learning. In *Proceedings of the Second International Conference on Communications and Networking in China (CHINACOM '07)*, pages 239–243. IEEE, Shanghai, China, August 2007. DOI: 10.1109/CHINACOM.2007.4469372.

[11] Yongli Ma, Zongjue Qian, Guochu Shou, and Yihong Hu. Study of Information Network Traffic Identification Based on C4.5 Algorithm. In *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, pages 1–5. IEEE, Dalian, China, October 2008. DOI: 10.1109/WiCom.2008.2678.

[12] Wei Li and Andrew W. Moore. A Machine Learning Approach for Efficient Traffic Classification. In *Proceedings of the Fifteenth IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'07)*, pages 310–317. IEEE, Istanbul, Turkey, October 2007. DOI: 10.1109/MASCOTS.2007.2.

[13] Samuel A. Moore, Daniel M. D'addario, James Kurinskas, and Gary M. Weiss. Are Decision Trees Always Greener on the Open (Source) Side of the Fence? In *Proceedings of the 5th International Conference on Data Mining (DMIN'2009)*, pages 185–188. The Lancaster Centre for Forecasting, Las Vegas, Nevada, USA, July 2009.

[14] Jason But, Philip Branch, and Tung Le. Rapid identification of BitTorrent Traffic. In *Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN)*, pages 536–543. IEEE, Denver, Colorado, USA, October 2010. DOI: 10.1109/LCN.2010.5735770.

[15] De Sensi, Daniele and Danelutto, Marco and Deri, Luca. Dpi over commodity hardware: implementation of a scalable framework using fastflow. Master's thesis, Università di Pisa, Italy, 2012. Accessible: `http://etd.adm.unipi.it/t/etd-02042013-101033/`.

[16] Silvio Valenti, Dario Rossi, Alberto Dainotti, Antonio Pescapè, Alessandro Finamore, and Marco Mellia. Reviewing Traffic Classification. In *Data Traffic Monitoring and Analysis*, pages 123–147. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-36784-7_6.

[17] Hyunchul Kim, Kimberly C. Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In *Proceedings of the 2008 ACM CoNEXT conference*, page 8. ACM New York, Madrid, Spain, December 2008. DOI: 10.1145/1544012.1544023.

[18] Kensuke Fukuda. Difficulties of identifying application type in backbone traffic. In *2010 International Conference on Network and Service Management (CNSM)*, pages 358–361. IEEE, Niagara Falls, Ontario, Canada, October 2010. DOI: 10.1109/CNSM.2010.5691234.

[19] Valentín Carela-Español, Pere Barlet-Ros, Alberto Cabellos-Aparicio, and Josep Solé-Pareta. Analysis of the impact of sampling on NetFlow traffic classification. *Computer Networks*, 55:1083–1099, 2011. DOI: 10.1016/j.comnet.2010.11.002.

[20] Shane Alcock and Richard Nelson. Libprotoident: Traffic Classification Using Lightweight Packet Inspection. Technical report, University of Waikato, August 2012. Accessible: `http://www.wand.net.nz/publications/lpireport`.

[21] Alberto Dainotti, Francesco Gargiulo, Ludmila I. Kuncheva, Antonio Pescapè, and Carlo Sansone. Identification of traffic flows hiding behind TCP port 80. In *IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, Cape Town, South Africa, May 2010. DOI: 10.1109/ICC.2010.5502266.

[22] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, et al. Transport layer identification of P2P traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134. ACM New York, Taormina, Sicily, Italy, August 2004. DOI: 10.1145/1028788.1028804.

[23] Andrew W. Moore and Konstantina Papagiannaki. Toward the Accurate Identification of Network Applications. In *Proceedings of the 6th International Conference on Passive and Active network Measurement PAM 2005*, pages 41–54. Springer Berlin Heidelberg, Boston, Massachusetts, USA, March 2005. DOI: 10.1007/978-3-540-31966-5_4.

[24] Francesco Gringoli, Luca Salgarelli, Maurizio Dusi, Niccolo Cascarano, Fulvio Risso, and Kimberly C. Claffy. Gt: picking up the truth from the ground for internet traffic. *ACM SIGCOMM Computer Communication Review*, 39(5):12–18, 2009. DOI: 10.1145/1629607.1629610.

[25] Chaofan Shen and Leijun Huang. On detection accuracy of L7-filter and OpenDPI. In *2012 Third International Conference on Networking and Distributed Computing (ICNDC)*, pages 119–123. IEEE, Hangzhou, China, October 2012. DOI: 10.1109/ICNDC.2012.36.

[26] Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Quantifying the accuracy of the ground truth associated with Internet traffic traces. *Computer Networks*, 55(5):1158–1167, April 2011. DOI: 10.1016/j.comnet.2010.11.006.

[27] Tomasz Bujlow, Kartheepan Balachandran, Sara Ligaard Nørgaard Hald, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for research on the Internet traffic. *TELFOR Journal*, 4(1):2–7, September 2012. Accessible: http://journal.telfor.rs/Published/Vol4No1/Vol4No1.aspx.

[28] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification. Technical report, Department of Computer Architecture (DAC), Universitat Politècnica de Catalunya (UPC), June 2013. Accessible: https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2013,en.html.

[29] Jawad Khalife, Amjad Hajjar, and Jesús Díaz-Verdejo. Performance of OpenDPI in Identifying Sampled Network Traffic. *Journal of Networks*, 8(1):71–81, January 2013. DOI: 10.4304/jnw.8.1.71-81.

[30] Steffen Gebert, Rastin Pries, Daniel Schlosser, and Klaus Heck. Internet access traffic measurement and analysis. In *Proceedings of the 4th international conference on Traffic Monitoring and Analysis (TMA'12)*, pages 29–42. Springer Berlin Heidelberg, Vienna, Austria, March 2012. DOI: 10.1007/978-3-642-28534-9_3.

[31] Jawad Khalife, Amjad Hajjar, and Jesús Díaz-Verdejo. On the Performance of OpenDPI in Identifying P2P Truncated Flows. In *AP2PS 2011, The Third International Conference on Advances in P2P Systems*, pages 79–84. IARIA, Lisbon, Portugal, November 2011.

[32] Péter Megyesi and Sándor Molnár. Finding Typical Internet User Behaviors. In *Proceedings of the 18th EUNICE Conference on Information and Communications Technologies (EUNICE 2012): Information and Communication Technologies*, pages 321–327. Springer Berlin Heidelberg, Budapest, Hungary, August 2012. DOI: 10.1007/978-3-642-32808-4_29.

[33] Ryan Goss and Reinhardt Botha. Deep Packet Inspection – Fear of the Unknown. In *Information Security for South Africa (ISSA), 2010*, pages 1–5. IEEE, Sandton, Johannesburg, South Africa, August 2010. DOI: 10.1109/ISSA.2010.5588278.

[34] Thomas M. Chen and Lucia Hu. Internet Performance Monitoring. *Proceedings of the IEEE*, 90(9):1592–1603, September 2002. DOI: 10.1109/JPROC.2002.802006.

[35] LIRNEasia Broadband QoSE Benchmarking project, 2008. [Online]. Available: http://lirneasia.net/projects/2008-2010/indicators-continued/broadband-benchmarking-qos-20/.

[36] K. v. M. Naidu, Debmalya Panigrahi, and Rajeev Rastogi. Detecting Anomalies Using End-to-End Path Measurements. In *Proceedings of the 27th Conference on Computer Communications IEEE INFOCOM 2008*, pages 16–20. IEEE, Phoenix, Arizona, USA, April 2008. DOI: 10.1109/INFOCOM.2008.248.

[37] Aboagela Dogman, Reza Saatchi, and Samir Al-Khayatt. An Adaptive Statistical Sampling Technique for Computer Network Traffic. In *Proceedings of the 7th International Symposium on Communication Systems Networks and Digital Signal Processing (CSNDSP 2010)*, pages 479–483. IEEE, Newcastle upon Tyne, England, United Kingdom, July 2010.

[38] Baek-Young Choi, Jaesung Park, and Zhi-Li Zhang. Adaptive Random Sampling for Traffic Load Measurement. In *Proceedings of the IEEE International Conference on Communications (ICC'03)*, volume 3, pages 1552–1556. IEEE, Anchorage, Alaska, USA, May 2003. DOI: 10.1109/ICC.2003.1203863.

[39] Gerhard Haßlinger. Implications of Traffic Characteristics on Quality of Service in Broadband Multi Service Networks. In *Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)*, pages 196–204. IEEE, Rennes, France, September 2004. DOI: 10.1109/EURMIC.2004.1333372.

[40] Shao Liu, Mung Chiang, Mathias Jourdain, and Jin Li. Congestion Location Detection: Methodology, Algorithm, and Performance. In *Proceedings of the 17th International Workshop on Quality of Service (IWQoS 2009)*, pages 1–9. IEEE, Charleston, South Carolina, USA, July 2009. DOI: 10.1109/IWQoS.2009.5201404.

[41] 802.11e Amendment, 2005. [Online]. Available: http://standards.ieee.org/getieee802/download/802.11e-2005.pdf.

[42] Mathilde Benveniste. 'Tiered contention multiple access'(TCMA), a QoS-based distributed MAC protocol. In *The 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 2, pages 598–604. IEEE, Lisbona, Portugal, September 2002. DOI: 10.1109/PIMRC.2002.1047292.

[43] Eldad Perahia and Robert Stacey. *Next Generation Wireless LANs: 802.11 n and 802.11 ac*. Cambridge University Press, May 2013.

[44] Understanding Traffic Prioritization – Juniper Networks, 2010. [Online]. Available: http://www.juniper.net/techpubs/software/junos-security/junos-security10.1/junos-security-swconfig-wlan/topic-37924.html.

Paper I

Volunteer-Based System for Research on the Internet Traffic

Tomasz Bujlow, Kartheepan Balachandran, Sara Ligaard Nørgaard Hald, Tahir Riaz, and Jens Myrup Pedersen

Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark
{tbu, kba, slh, tahir, jens}@es.aau.dk

This paper is reprinted from the *TELFOR Journal*, volume 4, number 1, pages 2–7. September 2012.

Accessible: http://journal.telfor.rs/Published/Vol4No1/Vol4No1.aspx.

Abstract

To overcome the drawbacks of the existing methods for traffic classification (by ports, Deep Packet Inspection, statistical classification), a new system was developed, in which the data are collected and classified directly by clients installed on machines belonging to volunteers. Our approach combines the information obtained from the system sockets, the HTTP content types, and the data transmitted through network interfaces. It allows to group packets into flows and

associate them with particular applications or the types of service. This paper presents the design and implementation of our system, the testing phase and the obtained results. The performed threat assessment highlights potential security issues and proposes solutions in order to mitigate the risks. Furthermore, it proves that the system is feasible in terms of uptime and resource usage, assesses its performance and proposes future enhancements. We released the system under *The GNU General Public License v3.0* and published it as a SourceForge project called *Volunteer-Based System for Research on the Internet.*

Keywords

computer networks, data collecting, performance monitoring, volunteer-based system

1 Introduction

This journal paper is an extended and revised version of [1], which was presented at the 19th Telecommunications Forum TELFOR 2011.

Monitoring of the data flowing in the inter-network is usually done to investigate the usage of network resources, and to comply with the law, as in many countries the Internet Service Providers (ISPs) are obligated to register users' activity. Monitoring can be also made for scientific purposes, like creating realistic models of traffic and applications for simulations, and to obtain accurate training data for statistical traffic classifiers.

This paper focuses on the last approach. There are many existing methods to assign the packets in the network to a particular application, but none of them were capable of providing high-quality per-application statistics when working in high-speed networks. Classification by ports or Deep Packet Inspection (DPI) can provide sufficient results only for a limited number of applications, which use fixed port numbers or contain characteristic patterns in the payload. Therefore, we designed, built, and tested a system, which collects the data directly from the machines belonging to volunteers who contribute with the traffic data. For the particular parts of the system, we described the available and chosen solutions. Our objective was to show that the system is feasible in terms of resource usage, uptime, and providing valid results. The remainder of this paper describes the previous work related to this research and then focuses on the design and the new implementation of the volunteer-based system. Finally, it shows the results of 3-months system tests and proposes further enhancements.

2 Related Work

Most methods for traffic classification use the concept of a flow defined as a group of packets, which have the same end IP addresses, ports, and use the same transport layer protocol. Flows are bidirectional, so packets going from the local machine to the remote server and from the remote server to the local machine belong to the same flow. In [2], the authors proposed to collect the data by Wireshark while running one application per host at a time so that all the captured packets will correspond to that application. But this method requires each application, whose traffic characteristics have to be captured, to be installed on the host, which is run once for each application. This solution is slow and not scalable. Secondly, all operating systems usually have background processes such as DNS requests and responses, system or program upgrades. They can damage statistics of the application traffic.

A DPI solution using *L7-filter* and a statistical classification solution are proposed in [3]. Using DPI is much more convenient than the previous method, as it can examine the data in any point in the network. Unfortunately, existing DPI tools are not able to accurately classify traffic belonging to some applications like Skype (in this case *L7-filter* relies on statistical information instead of the real traffic patterns, giving some false positives and false negatives [4]). Obtaining the training data for statistical classification based on statistical classifiers will not give us high accuracy of the new classifier. The idea of using DPI for classification of the training data for Machine Learning Algorithms was used in [5]. Moreover, the DPI classification is quite slow and requires a lot of processing power [2, 6]. It relies on inspecting the user data and, therefore, privacy and confidentiality issues can appear [2]. Application signatures for every application must be created outside the system and kept up to date [2], which can be problematic. Encryption techniques in many cases make DPI impossible.

Using application ports [7, 8] is a very simple idea, widely used by network administrators to limit the traffic generated by worms and other unwanted applications. This method is very fast and it can be applied to almost all routers and layer-3 switches existing on the market. Besides its universality, it is very efficient to classify some protocols operating on fixed port numbers. Using it, however, gives very bad results in detection of protocols using dynamic port numbers, like P2P or Skype [2, 6, 9]. The second drawback is not less severe: many applications try to use well-known port numbers to be treated in the network with a priority.

3 Volunteer-Based System

We developed a system, which collects flows of Internet data traffic together with the information about the application associated with each flow. The prototype version was called *Volunteer-based Distributed Traffic Data Collection System* and its architecture was described and analyzed in [10] and [11]. The design and the implementation of the

prototype had numerous weaknesses and stability issues. Therefore, a new implementation of the system has been made, called Volunteer-Based System (VBS). The new, reimplemented version of VBS was released under *The GNU General Public License v3.0* and published as a SourceForge project. The website [12] contains a broad description of the project illustrated with screenshots, a roadmap, binary packages, source code in the Git repository, comprehensive documentation of the source code, and a system for bug tracking and feature requests.

Both the prototype and VBS were developed in Java, using Eclipse environment, and that resulted in a cross-platform solution. Currently, only Microsoft Windows (XP and newer) and Linux are supported because of third-party libraries and helper applications used in the development. The system consists of clients installed on volunteers' computers and of a server responsible for storing the collected data.

The task of the client is to register the information about each data packet passing the Network Interface Card (NIC). Captured packets are categorized into flows, with the exception of traffic to and from the local network (file transfers between local peers are filtered out). The following attributes of the flow are captured: start and end times of the flow, number of packets contained by the flow, local and remote IP addresses, local and remote ports, transport layer protocol, name of the application, and name of the client associated with the flow. The client also collects information about all the packets associated with each flow: direction, size, TCP flags, and relative timestamp to the previous packet in the flow. One transport-layer flow can contain multiple application-layer streams of HTTP data, and each of them can carry different kinds of content, such as audio or video. For that reason, packets belonging to flows which contain HTTP content require additional information to be collected. Therefore, in this case, we additionally store the information about the content type for each packet of the flow. In fact, the information about the content type is present only in the first packet of the response made to an HTTP request. It means that for each HTTP request we have one packet containing the information about the content type, which allows us to logically split all the application-layer HTTP streams. The collected data are periodically transmitted to the server, which stores all the data for further analysis. The client consists of 4 modules running as separate threads: packet capturer, socket monitor, flow generator, and data transmitter.

Both the VBS client and the VBS server are designed to run in the background and to start automatically together with the operating system (as a Windows service or a Linux daemon). The prototype uses the free community version of Tanuki Java Service Wrapper [13], which provides support only for 32-bit JVMs, and which requires special packaging of the Java application and placement of the libraries. To avoid these limitations, it has been replaced with YAJSW [14], an open-source project that provides support for both 32-bit and 64-bit versions of Windows and Linux.

We implemented a fully automatic update system for VBS clients. To simplify the update process, we introduced three different version numbers in our software; the first

one is associated with the client, the second one with the server, and the third one with the structure of the SQLite database used for exchanging the information between the client and the server. The update is stored on the server. While registering on the server, the client asks if any update is available, and downloads it if possible. The update is automatically installed by a script executed by YAJSW during the next restart of the VBS client.

3.1 Packet Capturer

External Java libraries for collecting packets from the network rely on the already installed Winpcap (on Windows) or libpcap (on Linux), which makes the operating system dependency issue transparent to the application. The Jpcap [15] library used in the prototype is not suitable for processing packets from high-speed links, because transfers with rates higher than 3 MB/s cause Java Virtual Machine (JVM) to crash. Moreover, the *loopPacket* and the *processPacket* functions are broken causing random JVM crashes, so the only possibility is to process the packets one by one using *getPacket* (this bug is fixed in a new project called Jpcapng [16] evolved from Jpcap). Jpcap has not been developed since 2007 and Jpcapng since 2010, so there is no chance to get the bugs corrected. Therefore, we chose jNetPcap [17] as it contains even more useful features than Jpcap offered, such as detecting and stripping different kinds of headers (data-link, IP, TCP, UDP, HTTP) in the processed packets. It allows the client to capture packets on all the interfaces, not only on the Ethernet ones like the prototype, where the client needed to know the number of stripped bytes. jNetPcap is also able to filter out the local subnet traffic on the Pcap level by compiling dynamically Pcap filters, which saves system memory and CPU power. Contrary to processing each packet separately by the prototype of VBS, we decided to use the native function of Winpcap or libpcap called *loopPacket*. It allowed to lower the usage of the resources consumed by the VBS client. It is worth mentioning that the Winpcap library tends to crash when the computer is placed into the standby mode, sleep mode, or hibernation. Therefore, the packet capturer needs to be continuously monitored and restarted in the case of a crash. A need to restart the capturer also appears when new interfaces are detected in the system, for example, when a network cable is connected, or the switch of a wireless card is enabled. If an IP address on an interface is changed, the packet capturer needs to be restarted as well to prevent the confusion with recognizing the local and remote IP addresses.

3.2 Socket Monitor

The socket monitor calls the external socket monitoring tool every second to ensure that even very short flows are registered. In the prototype, the built-in Windows or Linux Netstat was used, but it takes up to 20 seconds for Windows Netstat to display the output on some machines. We tried to solve this issue by using CurrPorts [18] instead of Netstat on Windows. Unfortunately, the only way to export the socket information was

to write it to a file on the hard disk. It resulted in poor performance due to excessive disk reads and writes when executing CurrPorts each second. Finally, we chose Tcpviewcon, a console version of TCPView [19]. Tcpviewcon displays the information about the sockets in the console in a Netstat-like view, which allows us to process this information in the same manner as using Netstat. Using the external tools brings some licensing issues. These third-party applications must not be redistributed along with the VBS, but they need to be downloaded by the installer on the users' computers after accepting their license agreements.

VBS monitors both TCP and UDP sockets, contrary to the prototype, which was able to handle only TCP sockets. TCP sockets include the information about both endpoints (local and remote) because a connection is established, while UDP sockets only provide the information about the local host. Since only one application can listen on a given UDP port at a time, the information about the local IP address and the local port are fully sufficient to obtain the application name for the given flow. Nevertheless, it is not possible to use the information about the UDP socket to terminate the flow, because many flows to different remote points can coexist using one UDP socket. Therefore, UDP flows are always closed based on timeout. TCP sockets are created on a one-per-connection basis, so it is possible to precisely assign a socket to a flow and close the flow when the matching socket is closed.

3.3 Flow Generator

Collected packets are grouped into flows. If the application name can be received from the matching socket, it is assigned to the flow. When the flow is closed (the matching socket is closed or the flow is timed out in case the flow is not mapped to any socket), it is stored in the memory buffer. The prototype treated the flow and the packet data as raw byte arrays and stored them as binary files. However, it was impossible to detect the corruption of files or to look into the file to see what went wrong without binary file parsers. Therefore, we decided to use SQLite [20], which uses the proper data types (like integer, double, string) for all the captured information.

3.4 Data Transmitter

Before transmitting the data, the client authenticates itself to the server using a hard-coded plain-text password and obtains an identifier. The communication between the clients and the server uses raw sockets. The node authentication and the data transmission require separate connections between the clients and the server. When a sufficient number of flows are stored in the local database (the database exceeds 700 kB), the SQLite database file is transmitted and stored on the VBS server. The transmitted database file also includes the client identifier and the information about the operating system installed on the client machine.

3.5 Implementation of the Server

The prototype server was based on threads. It received binary files from the clients and stored them in a separate directory for each client. The VBS server is also based on threads, however, it stores the collected data differently. The first thread authenticates the clients and assigns identifiers to them. The second thread receives files from clients and stores them in a common folder, which is periodically checked by the third thread. The files are checked for corruption and the proper SQLite database format, then they are extracted into the database. A synchronization method is used to avoid a situation where the third thread tries to process a file, which was not transferred completely – the extension of the file is changed after the file transfer is successful. The server uses the community edition of MySQL as the database, as it is quite fast and reliable for storing significant amounts of data.

4 Testing Phase

The system was implemented and tested over a period of 3 months, to test its feasibility and usefulness in collecting valid data. The server was installed at Aalborg University on a PC equipped with an Intel Core i3 550 / 3.2 GHz processor, 4 GB of DDR3 SDRAM memory, and 70 GB hard disk and using Ubuntu 11.04 as OS. The clients were installed on 4 computers placed in private houses in Denmark and in Poland as well as on 23 machines installed in computer classrooms in *Gimnazjum nr 3 z Oddziałami Integracyjnymi i Dwujęzycznymi imienia Karola Wojtyły w Mysłowicach*, a high school in Poland. The computers used for the test were equipped with various hardware and operating systems. The objective was to prove that the system has high uptime, collects data from remotely located clients, and does not consume too much resources. The CPU usage by the VBS fluctuates with the average of around 1.7 % depending on the current rate of the traffic in the network. The CPU usage on the computers participating in our tests is shown in Figure 1. To avoid the complexity of illustrating the CPU usage over the long time of the experiment, we illustrated the occurrence rate of the CPU consumption by each client. As it is shown, the CPU consumption in most cases amounts to 5 % or less, while the consumption of 10 % or more is extremely rare. During the experiment, no JVM errors about exceeding the default allocated memory size occurred, so we assume that VBS is free of memory leaks. The average memory usage on all the tested machines was below 5 % of the installed system memory. The minimum required amount of system memory is 64 MB because of the requirements of the Java service wrapper YAJSW. Disk space usage varied depending on the scheduling.

The test results were obtained during around 3 months and in this time the clients analyzed 325.88 GB of Internet traffic data (accumulated data from all clients). On the server side, 22.8 GB of statistical data were collected, consisting of 0.9 GB of flows data (5,799,207 records), and 21.9 GB of packets data (446,987,507 records). The communi-

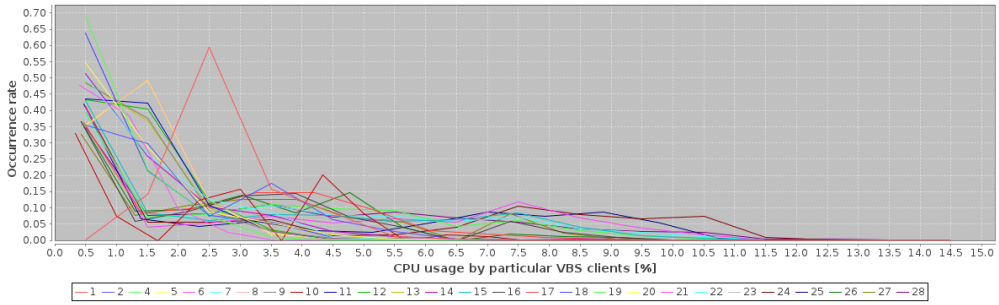

Figure 1: The CPU Usage by VBS Clients on Test Computers

cation between the VBS client and the server passes the network adapter as an ordinary remote connection, so it also appears in the database and is a subject to be included in the classification. During the test period, 4.21 % of the collected data correspond to the communication between the VBS client and the VBS server. Short flows, which contain less than 20 packets, represent 87 % of total number of flows. Around 12 % of flows were collected without the associated application name. TCP flows without the application name contain 13 packets in average comparing to the flows with the application name containing 74 packets in average. It means that the matching sockets for short flows were not registered by the socket monitor due to very short opening time. This rule does not apply to UDP flows because all the flows created by one application use the same UDP socket in the system. It means that either all or none flows associated with the socket have assigned an application name, regardless of the length of the particular flow. Detailed statistics on per-client basis are shown in Table 1.

An example of the stored flows data is shown in Table 2. The IP addresses were hidden for privacy reasons, the start and the end times of the flow (stored as Unix timestamps) were cut due to their length. This table also depicts a very interesting behavior of Skype – while the main voice stream is transmitted directly between the peers using UDP, there are plenty of TCP and UDP conversations with many different points (originally it was found to be around 50). The reason for this could be that the Skype user directory is decentralized and distributed among the clients.

Together with each flow, the information about every packet belonging to this flow is registered. One TCP conversation is presented in Table 3, some non-relevant packets are omitted to save space. This example shows that all the parameters are correctly collected. Thanks to such a detailed flow description, it can be used as a base for creating numerous different statistics. These precise statistics can be used as an input to Machine Learning Algorithms (MLAs). We tried this approach in [21] and we classified traffic belonging to 7 different applications with accuracy of over 99 %. In [22], we used VBS to distinguish different kinds of browser traffic, such as file download, web browsing,

Table 1: Statistics of the Operations of the VBS Clients

Client Id	Average CPU Usage [%]	Captured Traffic [GB]	Traffic Belonging to VBS [%]	Number of Captured Flows	UDP Flows [%]	Flows of Below 20 Packets [%]	Flows Without Application Names [%]
1	2.4	43.64	4.46	843552	32.00	85.57	61.28
2	1.1	0.04	2.04	149	0.00	65.77	23.49
4	0.5	248.95	4.10	3587227	51.42	86.12	3.05
5	1.0	29.09	4.75	1184464	46.82	92.56	1.58
6	0.8	0.08	4.90	1009	0.00	69.67	9.91
7	0.9	0.17	6.36	6943	8.44	76.51	13.38
8	0.9	0.34	10.39	16671	1.83	80.64	40.35
9	0.7	0.14	5.30	3084	0.29	71.07	18.00
10	2.2	0.02	5.31	774	0.52	73.00	19.12
11	0.7	0.19	6.82	4552	1.38	65.66	8.37
12	0.8	0.43	9.07	12000	0.77	80.19	27.79
13	0.8	0.11	6.81	4324	0.48	75.13	7.61
14	0.9	0.24	5.31	6186	1.00	69.95	9.80
15	2.6	0.04	11.51	5724	0.05	84.03	37.61
16	2.6	0.03	9.08	2227	0.27	75.12	23.53
17	2.3	0.08	9.39	5736	0.14	72.70	27.63
18	0.6	0.50	6.24	16158	0.03	67.81	13.67
19	2.6	0.44	9.04	28575	0.08	74.80	29.18
20	0.8	0.16	6.51	4905	0.20	75.37	14.82
21	3.2	0.39	8.01	22572	0.00	69.76	27.16
22	2.8	0.39	7.41	24389	0.00	72.58	31.77
23	0.4	0.19	4.95	5664	0.04	67.07	8.33
24	3.7	0.10	5.05	2692	0.00	73.59	33.51
25	3.5	0.06	3.20	1352	0.00	71.52	38.76
26	2.3	0.02	3.23	1056	0.00	69.22	12.69
27	2.7	0.04	5.74	1414	0.00	66.62	19.87
28	2.7	0.02	6.94	419	0.00	74.41	63.96

audio and video. The statistics obtained from the particular groups of the traffic were provided as an input to MLAs. Moreover, the presence of the packet size and the relative timestamp enables us to re-create characteristics of this traffic and, therefore, also the behavior of the application associated with the flow. The relative timestamp shows how much time passed from the previous packet in the flow.

Table 2: Example of the Stored Flows Data, the Timestamps are Stripped Due to Their Length

Flow Id	Client Id	Start Time	End Time	Number of Packets	Local IP	Remote IP	Local Port	Remote Port	Protocol Name	Socket Name
1	1	13...	13...	40	192.x.x.x	213.x.x.x	1133	110	TCP	thebat.exe
6	1	13...	13...	10	192.x.x.x	74.x.x.x	1151	80	TCP	opera.exe
7	1	13...	13...	10	192.x.x.x	91.x.x.x	1138	80	TCP	opera.exe
46012	1	13...	13...	20	192.x.x.x	85.x.x.x	23399	45527	TCP	Skype.exe
46013	1	13...	13...	20	192.x.x.x	78.x.x.x	23399	3598	TCP	Skype.exe
46014	1	13...	13...	11	192.x.x.x	41.x.x.x	23399	10050	TCP	Skype.exe
46015	1	13...	13...	15	192.x.x.x	41.x.x.x	23399	10051	TCP	Skype.exe
46016	1	13...	13...	207457	192.x.x.x	62.x.x.x	23399	14471	UDP	Skype.exe
46021	1	13...	13...	3	192.x.x.x	183.x.x.x	23399	33033	UDP	Skype.exe

Table 3: Chosen Packets from One Stored TCP Communication

Flow Id	Direction	Packet Size	SYN	ACK	PSH	FIN	RST	CWR	ECN	URG	Relative Timestamp [μs]	Content Type Id
18	OUT	48	1	0	0	0	0	0	0	0	0	1
18	IN	48	1	1	0	0	0	0	0	0	134160	1
18	OUT	40	0	1	0	0	0	0	0	0	200	1
18	OUT	105	0	1	1	0	0	0	0	0	20262	1
18	IN	77	0	1	1	0	0	0	0	0	79654	1
18	OUT	43	0	1	1	0	0	0	0	0	1651	1
18	IN	58	0	1	1	0	0	0	0	0	66950	1
18	OUT	40	0	1	0	0	0	0	0	0	175472	1
18	OUT	40	0	1	0	1	0	0	0	0	1031011	1
18	IN	40	0	1	0	0	0	0	0	0	69279	1
18	IN	40	0	1	0	1	0	0	0	0	250	1
18	OUT	40	0	1	0	0	0	0	0	0	25	1

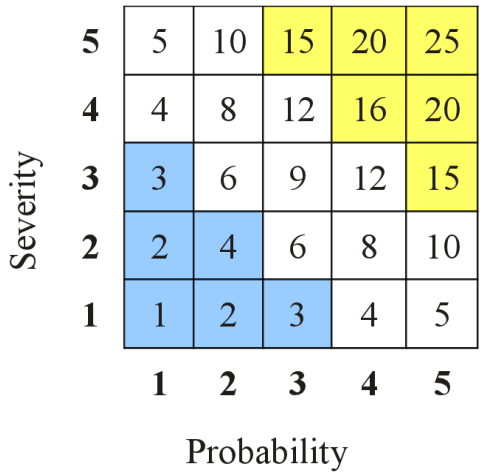

Figure 2: The Impact Matrix [23]

5 Threat Assessment

The Volunteer-Based System, as other distributed systems, is prone to various types of security attacks. For that reason, we made deep analysis of potential risks, and we suggested the solutions for mitigating or avoiding them. We started the assessment from finding all interfaces which can be used to interact with the system. These interfaces result from both the architecture of VBS and the use cases. Then, we composed the list of possible threats for each interface. For each threat, we assessed its probability and severity using the scale from 1 to 5, where 1 means the lowest probability or severity, and 5 means the highest. Table 4 shows the disclosed threats and the assigned values.

The decision about which threats need to be handled is based on the impact matrix shown in Figure 2 [23]. If the intersection of the probability and the severity of the threat lies in the cells marked with the yellow color, the threat should be handled. However, if it lies in the cells marked with the blue color, the threat does not need be handled. Threats belonging to the cells marked with the white color can be handled, but it is not required. We can see that only one threat requires handling in our system – the server needs to be protected from an SQL injection attack. There are several other issues which we can mitigate or avoid. The physical access is protected by placing the server in a safe room with attested locks, and a firewall can be used to protect the system from malicious connections. The threats to the system can also be further reduced by stripping the data of IP addresses or by using encryption.

Table 4: The Security Leaks in VBS with Assigned Probability and Severity

Interface / Threat	Probability (1–5)	Severity (1–5)	Handle
The user interface			
A user can delete the local data storage	3	1	No
A user can pollute data in his local data storage	1	2	No
A user can destroy the VBS system	3	1	No
A user can modify the local data storage by adding SQL commands (SQL injection)	1	5	Maybe
An attacker can hijack user's computer and redirect the data to another machine	1	1	No
The server interface			
An attacker can get physical access to the server	1	5	Maybe
The network interface			
An attacker can inject polluted data to the server	3	4	Maybe
An attacker can inject data containing SQL commands to the server (SQL injection)	4	5	Yes
An attacker can perform a Denial of Service attack on the server	3	2	Maybe
An attacker can hack the server and modify or delete the database	2	5	Maybe
An attacker can hack the server and change the file used to upgrade clients, which can result in loosing all VBS clients and transferring them into bots in attacker's botnet	1	5	Maybe
The communication between the clients and the server			
An attacker can sniff the communication between the client and the server	1	3	No
An attacker can pollute the data being sent by the client to the server	1	2	No
An attacker can modify the data being sent from the client to the server by adding SQL commands (SQL injection)	1	5	Maybe

6 Conclusion

The paper presents a novel volunteer-based system for collecting network traffic data, which was implemented and tested on 27 volunteers during around 3 months. With relatively long testing, the system has shown to be feasible in terms of resource usage and uptime. The obtained results proved that the system is capable of providing valid detailed information about the characteristics of the network traffic. Therefore, we can use the system to create the profiles of traffic generated by different applications. VBS is a field for constant improvements. As we assessed in the previous section, sufficient security needs to be implemented in the system. If it is possible, we should avoid using external tools (Netstat, TCPView), but extract the information about open sockets

directly from the system API. We need to consider developing an intelligent transfer protocol, which will allow to negotiate link parameters and to schedule transfers in order to effectively use the capacity of the link. A user-friendly installer will be the next step to make the system easier to use by non-qualified users.

VBS requires a valid IPv4 address to listen on a network interface, but IPv6 is also planned to be supported. Another issue arises when an encapsulation, data tunneling or network file systems (like SAMBA, NFS) are used. Then, only the most outer IP and TCP/UDP headers are inspected. The next issue is the lack of the application name for short flows. Volunteers' privacy also must be protected in a better way, for example by avoiding to store IP addresses in a clear text. Another concern, due to privacy issues, is how to find a large enough group of participating volunteers to be able to receive data for all the relevant applications. This issue is not resolved so far, but we believe that it will be easier to convince the users to install the software if it can provide some useful information to the user, like statistics about the amount of traffic belonging to the particular groups of applications.

Finally, the collected data can be used to create an emulator of different applications, different groups of applications, the Internet traffic under certain conditions, or at selected points of the time.

References

[1] Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for classification of traffic in computer networks. In *Proceedings of the 19th Telecommunications Forum TELFOR 2011*, pages 210–213. IEEE, Belgrade, Serbia, November 2011. DOI: 10.1109/TELFOR.2011.6143528.

[2] Jun Li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. Real-time P2P traffic identification. In *Proceedings of the IEEE Global Telecommunications Conference (IEEE GLOBECOM 2008)*, pages 1–5. IEEE, New Orleans, Louisiana, USA, December 2008. DOI: 10.1109/GLOCOM.2008.ECP.475.

[3] Riyad Alshammari and A. Nur Zincir-Heywood. Unveiling Skype encrypted tunnels using GP. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, Barcelona, Spain, July 2010. DOI: 10.1109/CEC.2010.5586288.

[4] L7-filter Supported Protocols, 2012. [Online]. Available: `http://l7-filter.sourceforge.net/protocols`.

[5] Wei Li and Andrew W. Moore. A Machine Learning Approach for Efficient Traffic Classification. In *Proceedings of the Fifteenth IEEE International Symposium*

on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'07), pages 310–317. IEEE, Istanbul, Turkey, October 2007. DOI: 10.1109/MASCOTS.2007.2.

[6] Ying Zhang, Hongbo Wang, and Shiduan Cheng. A Method for Real-Time Peer-to-Peer Traffic Classification Based on C4.5. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 1192–1195. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689126.

[7] Riyad Alshammari and A. Nur Zincir-Heywood. Machine Learning based encrypted traffic classification: identifying SSH and Skype. In *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA 2009)*, pages 1–8. IEEE, Ottawa, Ontario, Canada, July 2009. DOI: 10.1109/CISDA.2009.5356534.

[8] Sven Ubik and Petr Žejdl. Evaluating application-layer classification using a Machine Learning technique over different high speed networks. In *Proceedings of the Fifth International Conference on Systems and Networks Communications (ICSNC)*, pages 387–391. IEEE, Nice, France, August 2010. DOI: 10.1109/ICSNC.2010.66.

[9] Jing Cai, Zhibin Zhang, and Xinbo Song. An analysis of UDP traffic classification. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 116–119. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689203.

[10] Kartheepan Balachandran, Jacob Honoré Broberg, Kasper Revsbech, and Jens Myrup Pedersen. Volunteer-Based Distributed Traffic Data Collection System. In *Proceedings of the 12th International Conference on Advanced Communication Technology (ICACT 2010)*, volume 2, pages 1147–1152. IEEE, Phoenix Park, PyeongChang, Korea, February 2010.

[11] Kartheepan Balachandran and Jacob Honoré Broberg. Volunteer-Based Distributed Traffic Data Collection System. Master's thesis, Aalborg University, Department of Electronic Systems, Denmark, June 2010.

[12] Volunteer-Based System for Research on the Internet, 2012. [Online]. Available: `http://vbsi.sourceforge.net/`.

[13] Java Service Wrapper – Tanuki Software, 2011. [Online]. Available: `http://wrapper.tanukisoftware.com/doc/english/download.jsp`.

[14] YAJSW – Yet Another Java Service Wrapper, 2011. [Online]. Available: `http://yajsw.sourceforge.net/`.

[15] Jpcap – a Java library for capturing and sending network packets, 2007. [Online]. Available: http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/index.html.

[16] Jpcapng – fork of Jpcap, aka Jpcap 0.8, 2010. [Online]. Available: http://sourceforge.net/projects/jpcapng/.

[17] jNetPcap OpenSource | Protocol Analysis SDK, 2011. [Online]. Available: http://jnetpcap.com/.

[18] CurrPorts, Monitoring opened TCP/IP network ports / connections, 2011. [Online]. Available: http://www.nirsoft.net/utils/cports.html.

[19] TCPView for Windows, 2011. [Online]. Available: http://technet.microsoft.com/en-us/sysinternals/bb897437.

[20] SQLite, Self-contained, serverless, zero-configuration, transactional SQL database engine, 2011. [Online]. Available: http://www.sqlite.org/.

[21] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for classification of network traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of ICNC'12: 2012 International Conference on Computing, Networking and Communications (ICNC): Workshop on Computing, Networking and Communications*, pages 244–248. IEEE, Maui, Hawaii, USA, February 2012. DOI: 10.1109/ICCNC.2012.6167418.

[22] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. Classification of HTTP traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of the Fourth IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Web-based Service Architectures (PEDISWESA 2012)*, pages 882–887. IEEE, Cappadocia, Turkey, July 2012. DOI: 10.1109/ISCC.2012.6249413.

[23] Office of Government Commerce. *An Introduction to PRINCE2: Managing and Directing Successful Projects*. The Stationery Office (TSO), January 2009.

Paper II

A Method for Classification of Network Traffic Based on C5.0 Machine Learning Algorithm

Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen

Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark
{tbu, tahir, jens}@es.aau.dk

This paper is reprinted from the *Proceedings of ICNC'12: 2012 International Conference on Computing, Networking and Communications (ICNC): Workshop on Computing, Networking and Communications*, pages 244–248. IEEE, Maui, Hawaii, USA, February 2012.

DOI: 10.1109/ICCNC.2012.6167418.

Abstract

Monitoring of the network performance in a high-speed Internet infrastructure is a challenging task, as the requirements for the given quality level are service-dependent. Therefore, the backbone QoS monitoring and analysis in Multi-hop Networks requires the knowledge about the types of applications forming the current network traffic. To overcome the drawbacks of existing methods for traffic

classification, usage of C5.0 Machine Learning Algorithm (MLA) was proposed. On the basis of the statistical traffic information received from volunteers and C5.0 algorithm, we constructed a boosted classifier, which was shown to have the ability to distinguish between 7 different applications in the test set of 76,632–1,622,710 unknown cases with average accuracy of 99.3–99.9 %. This high accuracy was achieved by using high quality training data collected by our system, a unique set of parameters used for both training and classification, an algorithm for recognizing flow direction and the C5.0 itself. The classified applications include Skype, FTP, torrent, web browser traffic, web radio, interactive gaming and SSH. We performed subsequent tries using different sets of parameters and both training and classification options. This paper shows how we collected accurate traffic data, presents arguments used in classification process, introduces the C5.0 classifier and its options, and finally, evaluates and compares the obtained results.

Keywords

traffic classification, computer networks, C5.0, Machine Learning Algorithms (MLAs), performance monitoring

1 Introduction

One of the most important challenges in the network monitoring is how to measure the performance of high-speed Multi-hop Networks in a centralized manner. Each network carries data for numerous different kinds of applications, which have different performance requirements. Therefore, providing the information about the quality level requires the knowledge about what kinds of data are flowing in the network at the present time. Most of the current methods for traffic classification use the concept of a flow defined as a group of packets having the same end IP addresses, using the same transport protocol, and its port numbers. Flows are considered as bidirectional – packets going from the local machine to the remote server and from the remote server to the local machine are a part of the same flow.

Using application ports for traffic classification is a very simple idea widely used by network administrators to limit the traffic generated by worms and unwanted services. This method is very fast and can be applied to almost all the routers and layer-3 switches existing on the market. Apart from its universality, this method is very efficient to classify some protocols operating on fixed port numbers. Using it, however, gives very bad results in detection of protocols using dynamic port numbers, like P2P and Skype [1–3]. The second drawback is not less severe: many scam applications use well-known port numbers to be treated in the network with a priority. Deep Packet Inspection (DPI) solutions are quite slow and require a lot of processing power [1, 3]. Furthermore,

they rely on inspecting the user data and, therefore, privacy and confidentiality issues can appear [1]. Application signatures for every application must be created outside the system and kept up to date [1], which can be problematic. Worse, encryption techniques make DPI in many cases impossible.

Machine Learning Algorithms like K-Means, Naive Bayes Filter, C4.5, J48, Random Forests have much wider coverage. They can be used in any point of the network, providing very fast statistical detection of the application, to which the traffic belongs. Achievable detection rate correctness is over 95 % [1, 2, 4–9]. All the MLAs require significant amounts of training data for initial learning. The precision of the future classification by MLAs depends heavily on the quality of the training data. This paper introduces the usage of C5.0 in traffic classification and shows that this C4.5 successor is able to offer the classification accuracy of above 99 %.

The remainder of this document describes related previous work, gives an overview of our system, our method for collecting precise training data and isolating set of arguments used for classification, and then focuses on C5.0. The accuracy of the classification by C5.0 and the speed of generating the classifier was assessed when using various set of classification arguments and program options. Subsequently, the obtained results were presented and discussed.

2 Related Work

It was demonstrated in [1] that all the P2P applications behave similarly, so it is possible to use statistical analysis to detect even unknown applications. Several tries were made to classify accurately P2P and Skype traffic using older implementations of MLAs, like REPTree, C4.5, or J48. In [1], the authors proposed a few simple algorithms based on REPTree and C4.5, which are being able to classify P2P traffic using the first 5 packets of a flow. Their method based on C4.5 performed highly accurately (97 % of P2P traffic was classified properly), but the accuracy was not tested when starting packets from the flow were lost. Furthermore, the attribute set used for classification contained source and destination port numbers, what could make the classifier closely related to the current assignment of port numbers to particular applications in the training data.

Another approach to classify P2P applications was taken in [3] using a Java implementation of C4.5 called J48 to distinguish between 5 different applications. The authors tried to skip a number of packets in the beginning of the flow ranging from 10 to 1000 and they obtained only a little fluctuation in performance, with classification accuracy over 96 %. It was shown in [10] that the original C4.5 and J48 perform much different on relatively small or noisy data sets (the accuracy of J48 and C5.0 was in tested cases similar, and worse than C4.5). J48 processing using statistics based on sizes was implemented in [11] for detection of BitTorrent and FTP traffic, reaching the accuracy of around 98 %. This publication showed that behavior of data parameters contained in encrypted and unencrypted traffic generated by the same application looks

almost the same. Moreover, it was shown that zero-payload packets (ACK) can distort statistics based on sizes.

In [12], many different mechanisms of classification of the network traffic were evaluated, including C5.0. The achieved accuracy was around 88–97 % on traffic belonging to 14 different application classes. This not very high classification accuracy was probably partly due to preparing both training and test cases, where the decision attribute (application name) was obtained by DPIs (PACE, OpenDPI and L7-filter). These DPI solutions use multiple algorithms (including statistical analysis) to obtain the application name. Therefore, both training and test data were in some degrees inaccurate, which caused also more errors from the side of C5.0.

3 Overview of the Methods

In our research, the C5.0 classifier was intended to be a part of a system for Quality of Service (QoS) measurements in the core of the network [13]. The first task is to recruit volunteers from the users in the network, in which the system will be installed. The volunteers install on their computers a client program, which captures the relevant traffic information and submits the data to the server. On the server, these data are used to generate per-application traffic statistics. C5.0 Machine Learning Algorithm uses these statistics to learn how to distinguish between different types of applications and generate classification rules (decision trees). In our research, we focused on 7 different groups of applications instead of individual applications, because the QoS requirements within each group are similar (like for Firefox, Opera or Google Chrome web browsers).

The challenging task is to inspect nearly in real-time significant amount of traffic in the core of high-speed networks. Such systems deal with huge amounts of data and, therefore, only selected flows can be inspected due to memory and processing power limitations for quality assessment. Inspecting one or few flows per user at a time is enough, since when a user experiences problems, they usually concern all user's network activity. For better adjustment to applications used in different networks, the classifier was designed to be network-dependent, so it should be trained in each network independently. When the relevant flows are captured, per-flow statistics need to be generated. There are two kind of statistics generated at this step: used to determine the kind of application associated to that flow, and used to assess the QoS level in a passive way. The system uses the classification rules previously generated by C5.0 together with the first type of statistics to find out to which application the flow belongs. Then, on the basis of the kind of the application, the system determines acceptable ranges of values of the relevant QoS parameters. The last step is to check if the current values (obtained from flow statistics or in an active way) match the expected ones. If not, the quality of the given service is considered as degraded.

The subsequent paragraphs contain detailed description of our methods regarding:

- Collecting accurate training and test data by our Volunteer-Based System
- Criteria for the data flows used in our experiment
- Processing the flows and extracting the statistics
- Defining sets of classification arguments
- Assessing the accuracy of C5.0 while using various classification options

4 Obtaining the Data

A good solution for obtaining accurate training data can rely on collecting the flows at the user side along with the name of the associated application. We did this using our Volunteer-Based System. The basic idea and the design of the system were described in [14] and our current implementation in [15]. The system consists of clients installed on users' computers, and a server responsible for storing the collected data. The task of the client is to register the information about each flow passing the Network Interface Card (NIC), with the exception of traffic to and from the local subnet, to prevent capturing transfers between local peers. The following flow attributes were captured: start and end time of the flow, number of packets, local and remote IP addresses, local and remote ports, transport protocol, name of the application and client, which the flow belongs to. Apart from the information on the flow itself, the client also collected information about all the packets associated with each flow. These packet parameters were: direction, size, TCP flags, and relative timestamp to the previous packet in the flow. Information was then transmitted to the server, which stored all the data for further analysis in a MySQL database.

Another small software was developed for generating training and test files for the C5.0 classifier from the collected data. The following application groups were isolated: Skype main voice flow, FTP transfers (both uploads and downloads), torrent transfers, web browser traffic (except web radio), web radio traffic, the interactive game America's Army and SSH traffic. The requirements needed to be fulfilled by the traffic flows associated with each group are specified in Table 1.

Because of dynamic switching between the flows, the method had to be able to inspect a flow starting from any time point. For performance reasons, it is not possible to store all the flows in the database and to start inspection of the chosen one from its beginning. So, it had to be possible to assess the flow on the basis of the given number of packets or seconds from the middle of that flow. We assumed that the flow characteristics based on packet sizes within a network are independent of current conditions, contrary to the flow characteristics based on time parameters (which change quickly during e.g. congestion). Therefore, our method used the concept of a probe equals to a particular number of captured packets instead of particular number of seconds. The

Table 1: Requirements for Different Flows

Group	Requirement
Skype	protocol name = 'UDP' AND application name = 'Skype' AND no. of packets $\geq$ 200
FTP	application name = 'Filezilla'
Torrent	application name = 'uTorrent'
Web	(application name = 'firefox' OR application name = 'chrome' OR application name = 'opera') AND remote IP $\neq$ '195.184.101.203'
Web radio	client id = 3 AND application name = 'chrome' AND remote IP = '195.184.101.203'
Game	application name = 'AA3Game' AND protocol name = 'UDP'
SSH	application name = 'Putty'

probability of catching initial packets of each flow is very low due to dynamic switching between the flows. Moreover, count and size characteristics are different for the initial and for the remaining packets in the flow. To not disturb the accuracy of the classifier by statistics obtained from initial packets, we decided to ignore in the experiment ten initial packets of each flow, even if they were captured and stored. This feature excluded from the experiment flows possessing less than 15 packets, but this limitation was reasonable, because the QoS performance measurements rely in our case on long flows.

The direction of the flow was recognized on the basis of proportions of inbound to outbound payload bytes of the classified flow – the higher value was always considered as belonging to the inbound traffic. This way, streams with asymmetric load were always classified in the same way and we avoided noise affecting the accuracy.

We needed to find out what number of packets from the flow is needed to perform accurate classification. Each flow was divided into X groups of Y packets, where Y depends on the current iteration (we tested our algorithm on groups of 5, 10, 15,..., 90 packets), and X is a count of obtained groups. The dependency between X and Y is evident: more packets in a group means less groups in total (and, therefore, less training cases), but higher accuracy of statistics creating each particular case. Obtained groups were divided into 2 disjoint sets used later to generate statistics.

5 Classification Attributes

Each group from these 2 disjoint sets was used to generate one (respectively training and testing) case for the classifier. This way we never used the same cases for both training and classification. The attributes were divided into 2 sets. Set A contained 32 general continuous attributes based only on packet count and sizes, plus the target attribute. All the size parameters were based on the real payload length, not the packet length. To improve the ability to classify the encrypted traffic, the outer TCP/IP (40 B) or UDP/IP (28 B) header was removed, leaving only the data part. This set of parameters

consists of:

- Number of inbound / outbound / total payload bytes in the sample
- Proportion of inbound to outbound data packets / payload bytes
- Mean, minimum, maximum first quartile, median, third quartile, standard deviation of inbound / outbound / total payload size in the probe
- Ratio of small inbound data packets containing 50 B payload or less to all inbound data packets
- Ratio of small outbound data packets containing 50 B payload or less to all outbound data packets
- Ratio of all small data packets containing 50 B payload or less to all data packets
- Ratio of large inbound data packets containing 1300 B payload or more to all inbound data packets
- Ratio of large outbound data packets containing 1300 B payload or more to all outbound data packets
- Ratio of all large data packets containing 1300 B payload or more to all data packets
- Application: skype, ftp, torrent, web, web_radio, game, ssh

Set B contained 10 protocol-dependent attributes:

- Transport protocol: TCP, UDP
- Local port: well-known, dynamic
- Remote port: well-known, dynamic
- Number of ACK / PSH flags for the inbound / outbound direction: continuous
- Proportion of inbound packets without payload to inbound packets: continuous
- Proportion of outbound packets without payload to outbound packets: continuous
- Proportion of packets without payload to all the packets: continuous

By dividing the classification attributes, the research demonstrated how much the accuracy of the classifier depends on including set B into the classification. It is worth mentioning that the attributes contained in set B are very general: no real port numbers were stored, but only the information if the number is below 1024 (well-known) or above (dynamic). This way, at first, we did not influence the speed of the classifier by dividing the cases for each local and remote port number (resulting in generating port-based classifier), but still we were able to provide general information about the application model: client-server or peer-to-peer. Secondly, services can use different port numbers, but still they should be classified correctly, as the servers in the client-server model usually use only well-known port numbers, and the clients use dynamic ones. By avoiding using the port numbers, our solution should be able to identify accurately also encrypted traffic. Zero-payload packets (ACK) were treated in a special way by assignment to their own classification argument.

6 C5.0-Based Classifier

The C5.0 algorithm is a new generation of Machine Learning Algorithms (MLAs) based on decision trees [16]. It means that the decision trees are built from the list of possible attributes and the set of training cases, and then the trees can be used to classify the subsequent sets of test cases. C5.0 was developed as an improved version of a well-known and widely used C4.5 classifier and it has several important advantages over its ancestor [17]. The generated rules are more accurate and the time used to generate them is lower (even around 360 times on some data sets). In C5.0, several new techniques were introduced:

- Boosting: several decision trees are generated and combined to improve the predictions
- Variable misclassification costs: that makes possible to avoid errors, which can result in a harm
- New attributes: dates, times, timestamps, ordered discrete attributes
- Values can be marked as missing or not applicable for particular cases
- Supports sampling and cross-validation

The C5.0 classifier contains a simple command-line interface, which was used by us to generate the decision trees, rules and finally test the classifier. In addition, free C source code for including C5.0 classifier in external applications is available on the website of C5.0. Detailed description of C5.0 and all its options and abilities is published in the tutorial [18].

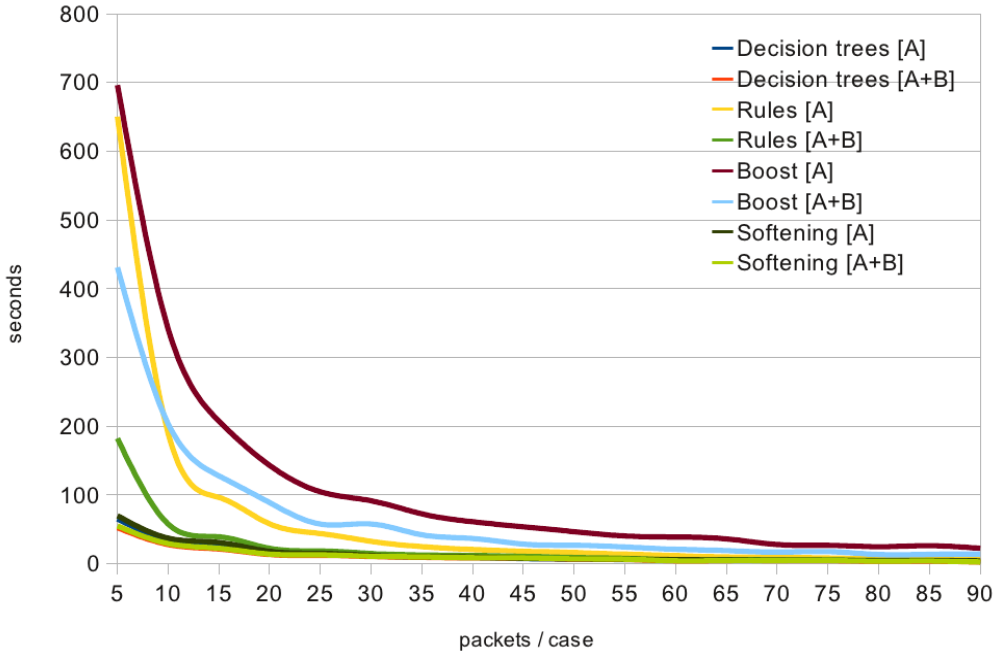

Figure 1: Time Spent for Generating the Classifier

7 Results

The training cases were provided to the C5.0 classifier to generate decision trees or classification rules. Then, the decision trees or the rules were used to classify the test cases. The experiment was repeated multiple times, each time using different sets of training and test cases (dependent on the number of packets used to create the case), different set of attributes used for classification (set A, or set A plus B), and different classification options (normal, rules generating, boosting, softening thresholds). We tested both the error rates of the provided classifiers (Table 2) and the time needed to construct them (Figure 1). The average error rates of the classifiers are shown in Figure 2 and the misclassification table for the classifier with the lower error rate (boosted classifier using both A and B sets of classification attributes, 75 packets used to construct each case) is presented in Figure 3. The experiment resulted in several important conclusions. First of all, using extended set of classification attributes (A + B) containing protocol-dependent attributes, we achieved lower bottom error rate (0.1 %) than using only size-based attributes from set A (2.7 %).

Table 2: Percent of Classification Errors When Using Different C5.0 Options

Number of Packets / Case	5	10	15	20	25	30	35	40	45
Number of Cases	1622710	791552	520661	384757	304388	251420	213810	185686	163715
Decision Trees [A]	5.0	4.0	3.5	3.2	3.2	3.0	3.1	2.9	2.9
Decision Trees [A+B]	0.5	0.4	0.7	0.5	0.5	1.0	0.5	0.9	0.3
Rules [A]	5.5	5.4	3.7	3.1	3.2	4.1	3.4	3.0	3.9
Rules [A+B]	0.5	0.4	0.6	0.4	0.4	1.0	0.3	0.3	0.3
Boost [A]	5.0	3.9	3.5	3.0	2.9	2.6	2.7	2.8	2.8
Boost [A+B]	0.4	0.4	0.3	0.4	0.4	0.3	0.2	0.2	0.2
Softening [A]	4.9	4.0	3.5	3.2	3.3	3.0	3.2	2.8	3.1
Softening [A+B]	0.5	0.4	0.7	0.5	0.5	0.9	0.5	0.9	0.3

Number of Packets / Case	50	55	60	65	70	75	80	85	90
Number of Cases	146248	131818	119710	109449	100908	93572	87225	81546	76632
Decision Trees [A]	3.0	2.9	3.0	3.2	3.0	3.0	3.0	3.1	3.0
Decision Trees [A+B]	0.4	0.5	0.4	0.3	0.4	0.4	0.3	0.3	0.3
Rules [A]	3.2	2.9	3.1	3.0	2.9	2.9	2.9	2.8	2.9
Rules [A+B]	0.4	0.2	0.4	0.3	0.2	0.2	0.3	0.2	0.3
Boost [A]	2.9	4.2	4.5	3.1	4.0	4.4	4.3	3.8	2.9
Boost [A+B]	0.7	0.2	0.2	0.2	0.2	0.1	0.2	0.3	0.2
Softening [A]	3.0	3.0	3.0	3.1	3.0	3.0	2.9	3.0	3.0
Softening [A+B]	0.4	0.5	0.4	0.3	0.4	0.4	0.3	0.3	0.3

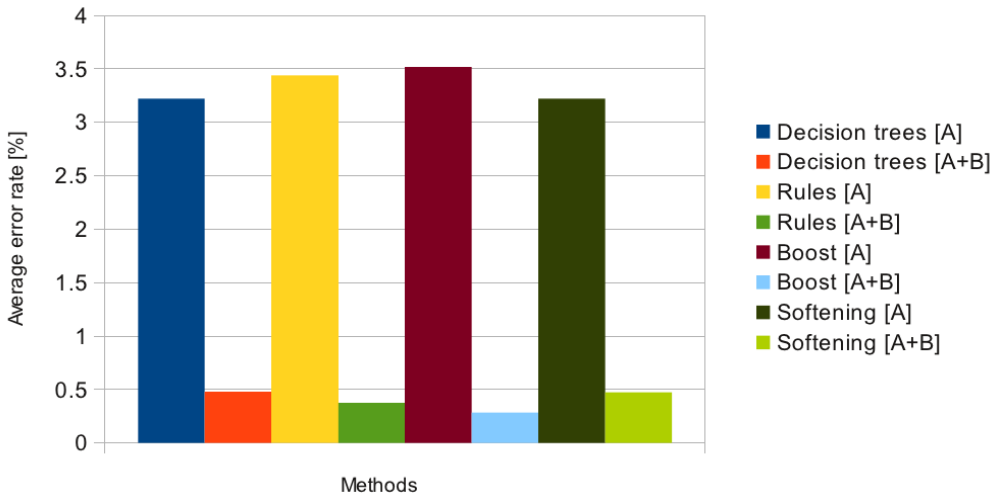

Figure 2: Average Error Rates of the Classifiers

The time used to construct classifiers from the extended set of attributes was also lower than when using only set A. Both these observations were completely independent of classification options. The lowest error rate of 0.1 % was achieved by using the boosted classifier, comparing to 0.3 % error rate when using the standard classification without any options. However, creating the boosted classifier took around 10 times more time than creating the standard classifier. Furthermore, our research demonstrated that creating the rules instead of decision trees, or using softened thresholds had no or only a little impact on the error rate, while it extended dramatically the time used to construct the classifier.

We also measured which way of training the classifier is the most optimal. The research showed that the classification error rate was the highest when using numerous cases, each created using statistics derived from 5 subsequent packets. The low precise statistics constructed from small number of packets were not sufficient to make the classifier accurate, even, if the count of them was significant. The classification error was decreasing while we were increasing the number of packets from which the statistics were generated, and it stabilized when we used 35 or more packets. Further increasing of the number of packets used to construct the case further did not improve the accuracy significantly, probably because it was compensated by a smaller number of provided training cases.

```
  (a)   (b)   (c)   (d)   (e)   (f)   (g)    <-classified as
 ----  ----  ----  ----  ----  ----  ----
  650                               3          (a): class skype
        196    68                             (b): class ftp
         19 83929     3                       (c): class torrent
                   6799                       (d): class web
                      8   388                 (e): class web_radio
   16           9                1419         (f): class game
                      7                  58   (g): class ssh
```

Figure 3: Misclassification Table, the Best Case

8 Conclusion

This paper presents a novel method based on C5.0 MLA for distinguishing different kinds of traffic in computer networks. It was demonstrated that our method is feasible to classify the traffic belonging to 7 different applications with average accuracy of 99.3–99.9 %, when using accurate data sets for both training and testing the boosted classifier. Our results proved that the classifier is able to distinguish traffic which appears to be similar, like web browser traffic and a radio streamed via a web page. The classifier did not have problems with distinguishing interactive traffic: Skype, game and SSH. We observed, however, that FTP and Torrent file transfers have very similar flow characteristics and, therefore, a significant number of packets were misclassified between these two classes. Our method is a field for more experiments and further improvements. In this research both the training and test data sets were disjoint, but collected from the same users. As the next step, we consider to involve numerous users to assess the accuracy using data sets obtained from different networks.

References

[1] Jun Li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. Real-time P2P traffic identification. In *Proceedings of the IEEE Global Telecommunications Conference (IEEE GLOBECOM 2008)*, pages 1–5. IEEE, New Orleans, Louisiana, USA, December 2008. DOI: 10.1109/GLOCOM.2008.ECP.475.

[2] Jing Cai, Zhibin Zhang, and Xinbo Song. An analysis of UDP traffic classification. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 116–119. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689203.

[3] Ying Zhang, Hongbo Wang, and Shiduan Cheng. A Method for Real-Time Peer-to-Peer Traffic Classification Based on C4.5. In *Proceedings of the 12th IEEE*

International Conference on Communication Technology (ICCT), pages 1192–1195. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689126.

[4] Riyad Alshammari and A. Nur Zincir-Heywood. Machine Learning based encrypted traffic classification: identifying SSH and Skype. In *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA 2009)*, pages 1–8. IEEE, Ottawa, Ontario, Canada, July 2009. DOI: 10.1109/CISDA.2009.5356534.

[5] Sven Ubik and Petr Žejdl. Evaluating application-layer classification using a Machine Learning technique over different high speed networks. In *Proceedings of the Fifth International Conference on Systems and Networks Communications (ICSNC)*, pages 387–391. IEEE, Nice, France, August 2010. DOI: 10.1109/ICSNC.2010.66.

[6] Riyad Alshammari and A. Nur Zincir-Heywood. Unveiling Skype encrypted tunnels using GP. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, Barcelona, Spain, July 2010. DOI: 10.1109/CEC.2010.5586288.

[7] Li Jun, Zhang Shunyi, Lu Yanqing, and Zhang Zailong. Internet Traffic Classification Using Machine Learning. In *Proceedings of the Second International Conference on Communications and Networking in China (CHINACOM '07)*, pages 239–243. IEEE, Shanghai, China, August 2007. DOI: 10.1109/CHINACOM.2007.4469372.

[8] Yongli Ma, Zongjue Qian, Guochu Shou, and Yihong Hu. Study of Information Network Traffic Identification Based on C4.5 Algorithm. In *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, pages 1–5. IEEE, Dalian, China, October 2008. DOI: 10.1109/WiCom.2008.2678.

[9] Wei Li and Andrew W. Moore. A Machine Learning Approach for Efficient Traffic Classification. In *Proceedings of the Fifteenth IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'07)*, pages 310–317. IEEE, Istanbul, Turkey, October 2007. DOI: 10.1109/MASCOTS.2007.2.

[10] Samuel A. Moore, Daniel M. D'addario, James Kurinskas, and Gary M. Weiss. Are Decision Trees Always Greener on the Open (Source) Side of the Fence? In *Proceedings of the 5th International Conference on Data Mining (DMIN'2009)*, pages 185–188. The Lancaster Centre for Forecasting, Las Vegas, Nevada, USA, July 2009.

[11] Jason But, Philip Branch, and Tung Le. Rapid identification of BitTorrent Traffic. In *Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN)*, pages 536–543. IEEE, Denver, Colorado, USA, October 2010. DOI: 10.1109/LCN.2010.5735770.

[12] De Sensi, Daniele and Danelutto, Marco and Deri, Luca. Dpi over commodity hardware: implementation of a scalable framework using fastflow. Master's thesis, Università di Pisa, Italy, 2012. Accessible: http://etd.adm.unipi.it/t/etd-02042013-101033/.

[13] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for Assessing Quality of Service in Broadband Networks. In *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT)*, pages 826–831. IEEE, Phoenix Park, PyeongChang, Korea, February 2012. Accessible: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6174795.

[14] Kartheepan Balachandran, Jacob Honoré Broberg, Kasper Revsbech, and Jens Myrup Pedersen. Volunteer-Based Distributed Traffic Data Collection System. In *Proceedings of the 12th International Conference on Advanced Communication Technology (ICACT 2010)*, volume 2, pages 1147–1152. IEEE, Phoenix Park, PyeongChang, Korea, February 2010.

[15] Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for classification of traffic in computer networks. In *Proceedings of the 19th Telecommunications Forum TELFOR 2011*, pages 210–213. IEEE, Belgrade, Serbia, November 2011. DOI: 10.1109/TELFOR.2011.6143528.

[16] Information on See5/C5.0 – RuleQuest Research Data Mining Tools, 2011. [Online]. Available: http://www.rulequest.com/see5-info.html.

[17] Is See5/C5.0 Better Than C4.5, 2009. [Online]. Available: http://www.rulequest.com/see5-comparison.html.

[18] C5.0: An Informal Tutorial, 2011. [Online]. Available: http://www.rulequest.com/see5-unix.html.

Paper III

A Method for Evaluation of Quality of Service in Computer Networks

Tomasz Bujlow, Sara Ligaard Nørgaard Hald, Tahir Riaz, and Jens Myrup Pedersen

Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark
{tbu, slh, tahir, jens}@es.aau.dk

This paper is reprinted from the *ICACT Transactions on the Advanced Communications Technology (TACT)*, volume 1, number 1, pages 17–25. Global IT Research Institute (GiRI), July 2012.

Accessible: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6488383.

Abstract

Monitoring of the Quality of Service (QoS) in high-speed Internet infrastructures is a challenging task. However, precise assessments must take into account the fact that the requirements for the given quality level are service-dependent. The backbone QoS monitoring and analysis requires processing of large amounts of

data and the knowledge about the kinds of applications, which generate the traffic. To overcome the drawbacks of existing methods for traffic classification, we proposed and evaluated a centralized solution based on the C5.0 Machine Learning Algorithm (MLA) and decision rules. The first task was to collect and to provide to C5.0 high-quality training data divided into groups, which correspond to different types of applications. It was found that the currently existing means of collecting data (classification by ports, Deep Packet Inspection, statistical classification, public data sources) are not sufficient and they do not comply with the required standards. We developed a new system to collect the training data, in which the major role is performed by volunteers. Client applications installed on volunteers' computers collect the detailed data about each flow passing through the network interface, together with the application name taken from the description of system sockets. This paper proposes a new method for measuring the level of Quality of Service in broadband networks. It is based on our Volunteer-Based System to collect the training data, Machine Learning Algorithms to generate the classification rules and the application-specific rules for assessing the QoS level. We combine both passive and active monitoring technologies. The paper evaluates different possibilities of the implementation, presents the current implementation of the particular parts of the system, their initial runs and the obtained results, highlighting parts relevant from the QoS point of view.

Keywords

broadband networks, data collecting, Machine Learning Algorithms, performance monitoring, Quality of Service, traffic classification, volunteer-based system

1 Introduction

This journal paper is an extended and revised version of [1], which was presented at the 14th International Conference on Advanced Communication Technology (ICACT 2012).

One of the most interesting challenges in today's world is how to measure the performance of computer network infrastructures, when different types of networks are merged together. In the last few years, the data-oriented networks evolved into converged structures, in which the real-time traffic, like voice calls or video conferences, is more and more important. The structure is composed of traditional data cable or more modern fiber links, existing Plain Old Telephone Service (POTS) lines used to provide analog services (voice telephony), or digital services (ADSL, PBX, ISDN), and nowadays also of mobile and wireless networks. There are numerous methods for the measurement of Quality of Service (QoS) in the current use, which provide the measurements both on

the user side and in the core of the network. Internet Service Providers are interested in centralized measurements and detecting problems with particular customers before the customers start complaining about the problems, and if possible, before the problems are even noticed by the customers.

Each network carries data for numerous different kinds of applications. QoS requirements are dependent on the service. The main service-specific parameters are bandwidth, delay, jitter, and packet loss. Regarding delay, we can distinguish strict real time constraints for voice and video conferences, and interactive services from delivery in relaxed time frame. In a conversation, delay of about 100 ms is hardly noticeable, but 250 ms of delay means an essential degradation of the transmission quality, and more than 400 ms is considered as severely disturbing [2].

Therefore, in order to provide detailed information about the quality level for the given service in the core of the network, we need to know, what kinds of data are flowing in the network at the present time. Processing all the packets flowing in a high-speed network and examining their payload to get the application name is a very hard task, involving large amounts of processing power and storage capacity. Furthermore, numerous privacy and confidentiality issues can arise. A solution for this problem can be the use of Machine Learning Algorithms (MLAs), which use previously generated decision rules, which are based on some statistical information about the traffic. In our research, we used one of the newest MLAs – C5.0. MLAs need very precise training sets to learn how to accurately classify the data, so the first issue to be solved was to find a way to collect high-quality training statistics.

In order to collect the necessary statistics and generate the training sets for C5.0, a new system was developed, in which the major role is performed by volunteers. Client applications installed on their computers collect the detailed information about each flow passing through the network interface, together with the application name taken from the description of the system sockets. The information about each packet belonging to the flow is also collected. Our volunteer-based system guarantees precise and detailed data sets about the network traffic. These data sets can be successfully used to generate statistics used as the input to train MLAs and to generate accurate decision rules.

The knowledge about the kind of application to which the traffic belongs obtained from MLAs can be used together with traffic requirements for the given application to assess the QoS level in the core of the real network. The real traffic needs to be sampled to obtain the necessary raw statistics. Parameters like jitter, burstiness, download and upload speed (and delay and packet loss for TCP traffic) can be assessed directly on the basis of the information obtained from the captured traffic. To assess the delay and packet loss for UDP traffic, active measurement techniques must be involved (like ping measurements in both directions).

The remainder of this document is split into several sections, which describe in detail the system architecture and some parts of the implementation. Section 2 contains the overview of the current methods for assessing the network QoS level. Both passive and

active methods are described along with their advantages and weaknesses. Section 3 gives the overview of our methods, so the reader is able to understand how the particular components are built and connected with each other. Section 4 describes the current methods used for traffic classification in computer networks and it explains why they are not sufficient for our needs. Section 5 presents our new tool used for collecting and classification of the network traffic – the Volunteer-Based System (VBS). Section 6 shows how the statistical parameters are obtained from the data collected by VBS. Section 7 evaluates different Machine Learning Algorithms and shows why we chose C5.0 to be included in our system. Section 8 demonstrates the design and implementation of the system, while Section 9 summarizes the most important points.

2 Related Work

During the last 20 years we have been witnesses to the subsequent and increasing growth of the global Internet and the network technology in general. The broadband and mobile broadband performance today is mainly measured and monitored by speed. However, there are several other parameters, which are important for critical business and real-time applications, such as voice and video applications or first-person shooter games. These parameters include round trip time, jitter, packet loss, and availability [3, 4].

The lack of the centralized administration makes it difficult to impose a common measurement infrastructure or protocol. For example, the deployment of active testing devices throughout the Internet would require a separate arrangement with each service provider [3]. This state of affairs led to some attempts to make simulation systems representing real characteristics of the traffic in the network. Routers and traffic analyzers provide passive single-point measurements. They do not measure the performance directly, but the traffic characteristics are strongly correlated with the performance. Routers and switches usually feature a capability to mirror incoming traffic to a specific port, where a traffic meter can be attached. The main difficulty in passive traffic monitoring is the steadily increasing rate of transmission links (10 or 100 GB/s), which can simply overwhelm routers or traffic analyzers, which try to process packets. It forces the introduction of packet sampling techniques and, therefore, it also introduces the possibility of inaccuracies. Even at 1 Gbit/s, the measurements can result in enormous amounts of data to process and store within the monitoring period [3].

To overcome the heavy load in the backbone and to not introduce inaccuracies, a smart monitoring algorithm was needed. There are several approaches to estimate which traffic flows need to be sampled. A path anomaly detection algorithm was proposed in [5]. The objective was to identify the paths, whose delay exceeds their threshold, without calculating delays for all paths. Path anomalies are typically rare events, and for the most part, the system operates normally, so there is no need to continuously compute delays for all the paths, wasting processor, memory, and storage resources [5]. Authors propose a sampling-based heuristic to compute a small set of paths to monitor,

reducing monitoring overhead by nearly 50 % comparing to monitoring all the existing paths.

The next proposals on how to sample network traffic in an efficient way were made on the basis of adaptive statistical sampling techniques, and they are presented in [6] and [7].

If a congestion is detected, from user's perspective it is very important to know, if the congestion is located on the local or on the remote side. If the link experiences a local congestion, the user may be able to perform certain actions, e.g. shut down an application, which consumes a lot of bandwidth, to ease the congestion. On the other hand, if the congested link is a remote link, either in the Internet core or at the server side, the back-off of the low-priority applications on the user's side is unnecessary. It only benefits the high-priority flows from other users, which compete for that link. Since this altruistic behavior is not desirable, the low priority TCP only needs to back off, when the congested link is local [8].

Detecting the location of the congestion is a challenging problem due to several reasons. First of all, we cannot send many probing packets, because it causes too much overhead, and it even expands the congestion. Secondly, without a router support, the only related signals to the end applications are packet losses and delays. If the packet losses were completely synchronized (packets were dropped from all the flows), the problem would be trivial. In the reality, the packet loss pattern is only partially synchronized [8]. Authors of [8] attempted to solve the problem of detecting the location of the congestion by using the synchronization of the behavior of loss and delay across multiple TCP sessions in the area controlled by the same local gateway. If many flows see a synchronized congestion, the local link is the congested link. If the congested link is remote, it is less likely that many flows from the same host pass the same congested link at the same time. If there is only a small number of flows which see the congestion, the authors performed an algorithm based on queuing delay patterns. If the local link is congested, most flows typically experience high delays at a similar level. Otherwise, the congestion is remote [8].

The traffic can be profiled according to the protocol composition. Usually, the predominance of the TCP traffic is observed (around 95 % of the traffic mix). When a congestion occurs, TCP sources respond by reducing their offered load, whereas UDP sources do not. It results in the higher ratio of UDP to TCP traffic. If the proportion becomes high and the bandwidth available to TCP connections becomes too low to maintain a reasonable transmission window, the packet loss increases dramatically (and TCP flows become dominated by retransmission timeouts) [3]. Packet sizes provide insight into the types of packets, e.g. short 40-44 bytes packets are usually TCP acknowledgments or TCP control segments (SYN, FIN or RST) [3].

Active methods for QoS monitoring raise three major concerns. First, the introduction of the test traffic will increase the network load, which can be viewed as an overhead cost for active methods. Second, the test traffic can affect measurements. Third, the

traffic entering an ISP can be considered as invasive and discarded or assigned to a low-priority class [3].

Within an administrative domain (but not across the entire Internet), the performance can be actively monitored using the data-link layer protocol below IP, as the Operations, Administration and Maintenance (OAM) procedure in ATM and MPLS networks. As a result, at the IP layer it is often desirable to measure performance using the IP/ICMP protocol. So far, most tools or methods are based on ping (ICMP echo request and echo reply messages) or traceroute (which exploits the TTL field in the header of the IP packet) [3].

Although the round-trip times measured by ping are important, ping is unable to measure the one-way delay without additional means like GPS to synchronize clocks at the source and destination hosts. Another difficulty is that pings are often discarded or low-prioritized by many ISP in their networks. Traceroute will not encounter this problem because UDP packets are used. However, traceroute has known limitations. For example, successive UDP packets sent by traceroute are not guaranteed to follow the same path. Also, the returned ICMP message may not follow the same path as the UDP packet that triggered it [3].

Although the end-to-end performance measurements can be carried out at the IP layer or the transport/application layer, the latest is capable of measurements closer to user's perspective. The basic idea is to run a program emulating a particular application that will send traffic through the Internet. All the parameters (delay, packet loss, throughput, etc) are measured on the test traffic. This approach has one major drawback - a custom software needs to be installed on the measurement hosts [3].

On the basis of the mentioned work, we found out that the existing solutions are not sufficient for precise QoS measurements. This state of affairs motivated us to create a new system which combines both passive and active measurement technologies.

3 Overview of the Methods

The flow chart of our system is shown in Figure 1. The following paragraphs contain the detailed description of our methods. At first, the volunteers must be recruited from the network users. The volunteers install on their computer a client program, which captures the relevant information about the traffic and submits the data to the server. On the server, these data are used to generate per-application traffic statistics. The C5.0 Machine Learning Algorithm uses these statistics to learn how to distinguish between different types of applications and, later, it generates the classification rules (decision trees).

In order to assess the network QoS level in the core of the network for particular users, we needed to find a method to capture the relevant traffic. The challenging task is to process significant amounts of traffic in high-speed networks. When the relevant flows are captured, per-flow statistics need to be generated. There are two kinds of statistics

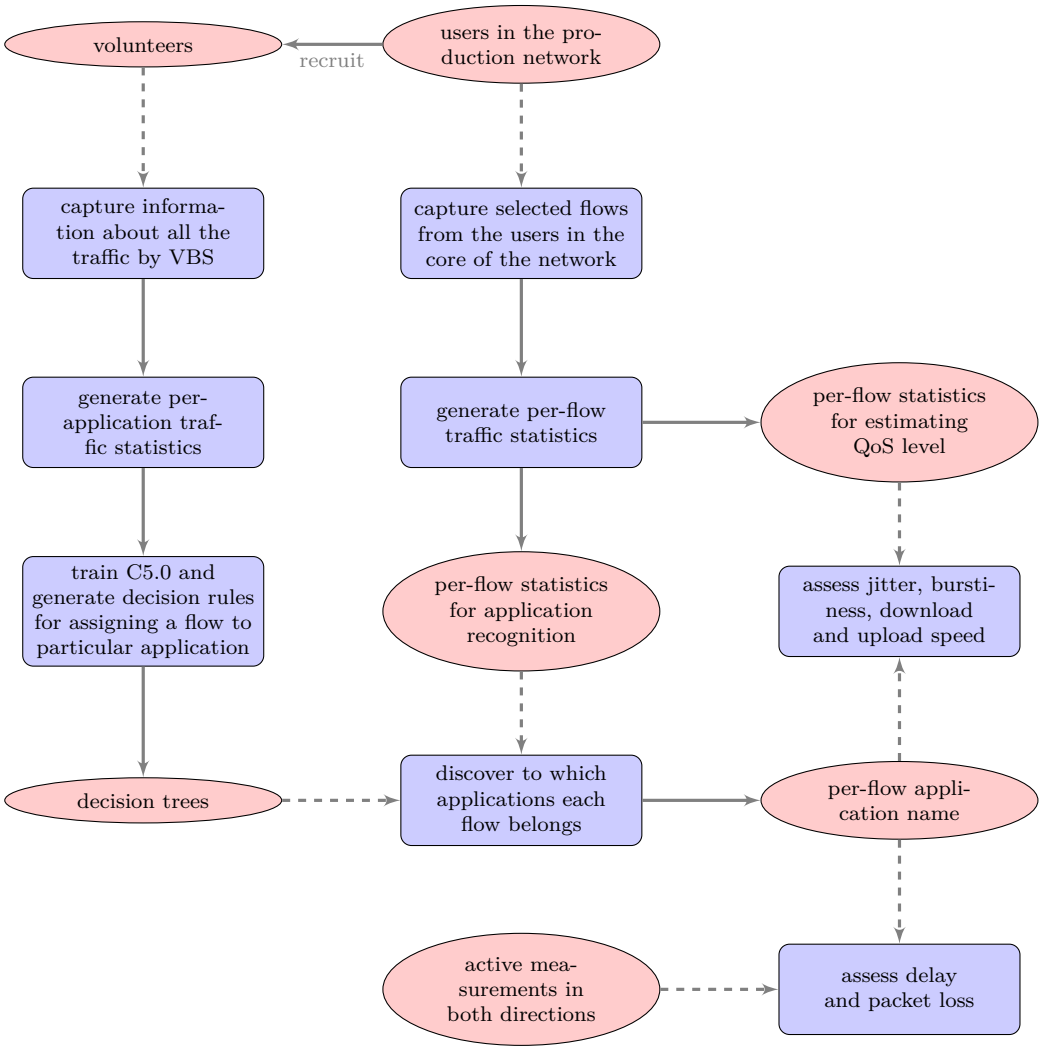

Figure 1: The Flow Chart of the System

generated at this step: one used for determining the kind of application associated with that flow, and one used for assessing the QoS level in the passive way. The system uses previously generated classification rules together with the first type of statistics to find out which application the flow belongs to. Then, on the basis of the kind of the application, the system determines the ranges of values of the relevant QoS parameters. The last step is to check if the current values (obtained from the flow statistics or in the active way) match the expected ones. If not, the quality of the given service is considered as degraded.

4 Current Methods for Obtaining Pre-Classified Data

There are many existing methods for obtaining pre-classified data, but none of them were feasible to deliver the data required by us to obtain accurate statistics, which could be used to train Machine Learning Algorithms (MLAs). The traffic classification requires the packets to be logically grouped into some structures, which could be assigned to the particular application. The most common used structure among the classification methods is the flow defined as a group of packets, which have the same end IP addresses, ports, and use the same transport layer protocol. In this section, we describe the methods and evaluate their usefulness in providing the data to our system.

4.1 Capturing Raw Data from the Network Interfaces

The first possibility is to install one application at a time on a host, and to capture its traffic by an external tool, such as Wireshark [9]. Unfortunately, this approach is very slow and it is not scalable. At first, it requires us to install on a host each application that generates the traffic we want to capture. Before installing the application, we must uninstall all other applications that can generate any network traffic. The next drawback is that every operating system has some background processes and many of them transmit some data through the network. An example of such a process is the system updater, which can run in background. There is no simple way to recognize packets belonging to the traffic generated by the application intentionally run by us, so the captured sample contains variable percentage of noise. Finally, some applications, for example, web browsers, can generate various types of traffic. Raw traffic capturers cannot distinguish interactive web traffic, web radio podcasts, video transmissions, or downloads of big files, performed by the same browser.

4.2 Classification by Ports

The port-based classification [10, 11] is very fast, and it is supported on almost all the layer-3 devices in computer networks. Unfortunately, this method is limited to services, protocols, and applications, which use fixed port numbers. It means that with

big probability we can correctly classify, for example, traffic generated by e-mail clients and file transfer clients using File Transfer Protocol (FTP), when they use the default ports to connect to servers. However, even in this case we have false positives and false negatives. False negatives result from non-standard ports used in this example by SMTP, POP3, or FTP servers. When a network administrator changes the port used by the given service (due to security reasons), the traffic is not classified correctly. False positives result from malicious applications, which intentionally use some well-known port numbers to be treated in the network with a priority, or to be able to transmit data at all. Such situation exists when a Torrent user runs his client on port 80, which causes the traffic to be treated as if it originated from a web server. Another big concern of port-based classification is the inability of recognizing different types of traffic using the same transport-layer protocol and the same transport-layer port. This drawback is strongly visible in the example of HTTP traffic, which can consist of data generated by interactive web browsing, audio and video streaming, file downloads, and HTTP tunneling for other protocols. Finally, the classification made by ports is unable to deal with protocols using dynamic port numbers, like BitTorrent or Skype [9, 12, 13].

4.3 Deep Packet Inspection (DPI)

The big advantage of the Deep Packet Inspection (DPI) [14] is the possibility to inspect the content of the traffic. It includes both inspecting particular packets, and inspecting flows in the network as the whole. For that reason, that makes possible to distinguish different kinds of content generated by the same application, or using the same application-layer protocol, such as HTTP. However, DPI is slow and requires a lot of processing power [9, 12]. Therefore, due to high load in today's network infrastructures, it is not feasible to run DPI in the core of the network. The speed of the Internet connections provided to private users tends to increase much faster than the processing power of their machines, so performing DPI on user's machines became impossible in our case. Feasibility to perform DPI on the user side does not depend only on possessing the necessary processing power, but also on the user's impression. High CPU usage tends to slow down the machine and it causes additional side-effects, for example, a howling CPU fan. For that reason, full DPI can be done only in a limited number of cases, namely on fast machines using a slow Internet connection. DPI also brings privacy and confidentiality issues, as it can reveal some highly sensitive personal data, such as information about used credit cards, logins and passwords, websites visited by the user, etc [9]. Moreover, DPI is unable to inspect encrypted traffic. Finally, DPI depends on signatures of various protocols, services, and applications, which need to be kept up to date.

4.4 Statistical Classification

Solutions using statistical classification became quite popular during the last few years [14]. To its characteristics we can include fast processing and low resource usage. Statistical classifiers are usually based on rules, which are automatically generated from samples of data. Therefore, such kinds of classifiers often make use of Machine Learning Algorithms (MLAs). Apart from all these advantages, statistical classifiers have one big drawback – they need to be trained on the samples of data. So the technique assumes that we have already correctly classified data, which we can provide as the input to train the statistical classifier. For that reason, we cannot use this method to collect and classify the initial portion of data.

5 Volunteer-Based System

The drawbacks of the existing methods for the classification of traffic in computer networks led us to the conclusion that we need to design and build another solution. Therefore, we decided to develop a system based on volunteers, which captures the traffic from their network interfaces, and groups the traffic into flows associated with the application name taken from Windows or Linux sockets. The architecture and the prototype were described and analyzed in [15] and [16], and the first version of our current implementation was presented in [17]. Afterwards, the system was extended to support recognizing different kinds of HTTP traffic, and it was named Volunteer-Based System (VBS). The detailed description and evaluation of the extended version of VBS can be found in [18]. We released the system under *The GNU General Public License v3.0*, and we published it as a SourceForge project. The project website [19] contains all the information needed to use the system (binary packages, screenshots, documentation and bug tracking system) as well as to perform further development (source code, roadmap, comprehensive documentation of the source code).

The architecture of the system is shown in Figure 2. This cross-platform solution consists of clients installed on users' computers (Microsoft Windows XP and newer and Linux are supported), and of a server responsible for storing the collected data. The client registers information about each flow passing the Network Interface Card (NIC), with the exception of the traffic to and from the local network. The captured information are: start time of the flow, anonymized identifiers of the local and the remote IP addresses, local and remote ports, transport layer protocol, anonymized identifier of the global IP address of the client, name of the application, and identifier of the client associated with the flow. The system also collects information about all the packets associated with each flow: identifier of the flow to which the packet belongs, direction, size, TCP flags, relative timestamp to the previous packet in the flow, and information about the HTTP content carried by the packet. It is worth mentioning that one flow can contain many higher-layer streams, for example, one TCP flow can contain multiple

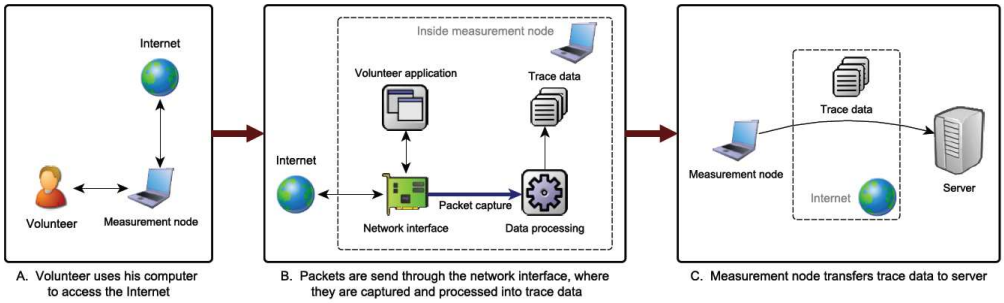

Figure 2: Overview of the VBS System [16]

HTTP conversations. Each of these conversations can transfer different kinds of content, like web pages, audio and video streams, or downloads of big files. For that reason, we extract from HTTP headers the information necessary to precisely separate the HTTP streams, and we append the information about the type of the stream to the first packet of the stream.

The collected information is then transmitted to the server, which stores all the data in a MySQL database for further analysis. The system was shown in [18] to be feasible and capable of providing detailed per-application information about the network traffic. An example of the stored flows on the server side is shown in Table 1. The IP addresses for privacy reasons are translated by a one-way hash function and they are stored as anonymized identifiers. The information about the packets belonging to one complete TCP conversation is presented in Table 2. As shown, this is an HTTP communication, during which there were transferred two files of the same type with identifier 22 (*text/html*).

The data collected during our experiments by the Volunteer-Based System were used for training the C5.0 Machine Learning Algorithm to be able to recognize the traffic generated by different types of applications and different types of traffic. The first approach, focusing on distinguishing 7 different applications (Skype, FTP, torrent, web browser, web radio, America's Army and SSH) and achieving accuracy of over 99 % was described and evaluated in [20]. The second approach, focusing on recognizing different kinds of HTTP content (audio, video, file downloads, interactive websites) was presented in [21].

Table 1: Example of the Stored Flows Data

Flow Id	Client Id	Start Time	Local IP	Remote IP	Local Port	Remote Port	Protocol Name	Global Client IP	Socket Name
1193430	4	1325445237826039	d1e0229	fb70266	48293	25395	UDP	178a02f1	uTorrent
2393417	5	1325445237826176	f4c025e	12230296	2276	80	TCP	177d02ef	chrome
1193423	1	1325445237826304	d20022b	11920285	53778	80	TCP	12350297	firefox
1484673	4	1325445237825884	d1e0229	12170293	58104	993	TCP	178a02f1	thebat
3429674	4	1325445236820017	d1e0229	14cb02b9	61159	80	TCP	178a02f1	Dropbox
3329860	1	1325445237044777	d20022b	1199028a	47801	80	TCP	12350297	plugin-con
3829589	1	1325445236797638	d20022b	124d0296	36868	80	TCP	12350297	wget
3474027	4	1325445212663601	d1e0229	14db02c2	63409	24536	UDP	178a02f1	Skype
4194793	1	1325445280781252	d20022b	1206028f	53331	22849	TCP	12350297	amule

Table 2: One TCP Communication Stored in the Database

Flow Id	Direction	Packet Size [B]	SYN Flag	ACK Flag	PSH Flag	FIN Flag	RST Flag	CWR Flag	ECN Flag	URG Flag	Relative Timestamp [μs]	Content Type Id
2784673	OUT	60	1	0	0	0	0	0	0	0	0	1
2784673	IN	60	1	1	0	0	0	0	0	0	30012	1
2784673	OUT	52	0	1	0	0	0	0	0	0	44	1
2784673	OUT	431	0	1	1	0	0	0	0	0	395	1
2784673	IN	52	0	1	0	0	0	0	0	0	30241	1
2784673	IN	527	0	1	1	0	0	0	0	0	2554	22
2784673	OUT	52	0	1	0	0	0	0	0	0	27	1
2784673	IN	539	0	1	1	0	0	0	0	0	10455	22
2784673	OUT	52	0	1	0	0	0	0	0	0	15	1
2784673	OUT	287	0	1	1	0	0	0	0	0	1383	1
2784673	OUT	52	0	1	0	1	0	0	0	0	15047	1
2784673	IN	269	0	1	1	0	0	0	0	0	16408	1
2784673	OUT	40	0	0	0	0	1	0	0	0	45	1
2784673	IN	52	0	1	0	1	0	0	0	0	13354	1
2784673	OUT	40	0	0	0	0	1	0	0	0	29	1

6 Obtaining Per-Application Statistics

The next step was to obtain the statistical profiles of flows for different applications. Therefore, we developed a tool for calculating statistics of several traffic attributes for each flow in the database, which fulfills our requirements. The statistics include 32 attributes based on sizes and 10 protocol-dependent attributes [20]. We suspect that the attributes based on sizes are independent of the current conditions in the network (like for example congestion). All the protocol-dependent attributes are very general. Precise port numbers are not used, but only the information about whether the port is well-known or dynamic. This way we avoid constructing a port-based classifier, but we can retain the information if the application model is more like client-server or peer-to-peer.

The general calculated statistics are [20]:

- Number of inbound / outbound / total payload bytes in the sample
- Proportion of inbound to outbound data packets / payload bytes
- Mean, minimum, maximum first quartile, median, third quartile, standard deviation of inbound / outbound / total payload size in the probe
- Ratio of small inbound data packets containing 50 B payload or less to all inbound data packets
- Ratio of small outbound data packets containing 50 B payload or less to all outbound data packets
- Ratio of all small data packets containing 50 B payload or less to all data packets
- Ratio of large inbound data packets containing 1300 B payload or more to all inbound data packets
- Ratio of large outbound data packets containing 1300 B payload or more to all outbound data packets
- Ratio of all large data packets containing 1300 B payload or more to all data packets
- Application: skype, ftp, torrent, web, web_radio, game, ssh

The protocol-dependent attributes are [20]:

- Transport protocol: TCP, UDP
- Local port: well-known, dynamic

- Remote port: well-known, dynamic
- Number of ACK / PSH flags for the inbound / outbound direction: continuous
- Proportion of inbound packets without payload to inbound packets: continuous
- Proportion of outbound packets without payload to outbound packets: continuous
- Proportion of packets without payload to all the packets: continuous

The precise process of obtaining these statistics was described in detail and evaluated in [20]:

7 Machine Learning Algorithms

In the recent literature, we can find numerous approaches to use Machine Learning Algorithms to classify the traffic in computer networks. The most widely used MLA classifiers are C4.5 [9] and its modified Java implementation called J48 [12, 22]. Based on statistical analysis, MLAs have the ability to assign a particular class (like P2P) even to traffic generated by unknown applications [9]. It was also proved in [22] that the statistical parameters for the encrypted and unencrypted traffic produced by the same application are similar and, therefore, the encrypted payload does not influence results of the training or the classification. The accuracy of the classification by MLAs was claimed to be over 95 % [9–11, 13, 14, 23–25]. The analysis of the related work can be found in [20].

It was found in [11] that the results of the classification are most accurate when the classifier was trained in the same network as the classification process was performed. This may be due to different parameters, which are constant in the particular network, but which differ among various networks. A good example is the Maximum Transmission Unit, which can easily influence statistics based on sizes. Therefore, in our design, we decided to train the classifier by volunteers in the same network as the classifier will be installed. This allows us to make a self-learning system, where a group of volunteers in the network delivers the data used for training the classifier constantly improving its accuracy, while all the users can be monitored in the core using the generated decision rules. The next advantage of the design is that even if some network users cannot participate in the data collecting process because of using other operating systems or devices than supported (like MacOS, Apple or Android smartphones), they will still be able to be monitored in the core of the network because of rules created on the basis of the data collected from the other users.

Our system uses the C5.0 MLA, which is a successor of C4.5. It is proven to have many advantages over its predecessor, such as higher accuracy, possibilities to use boosting, pruning, weighting and winnowing attributes. Furthermore, the time needed to

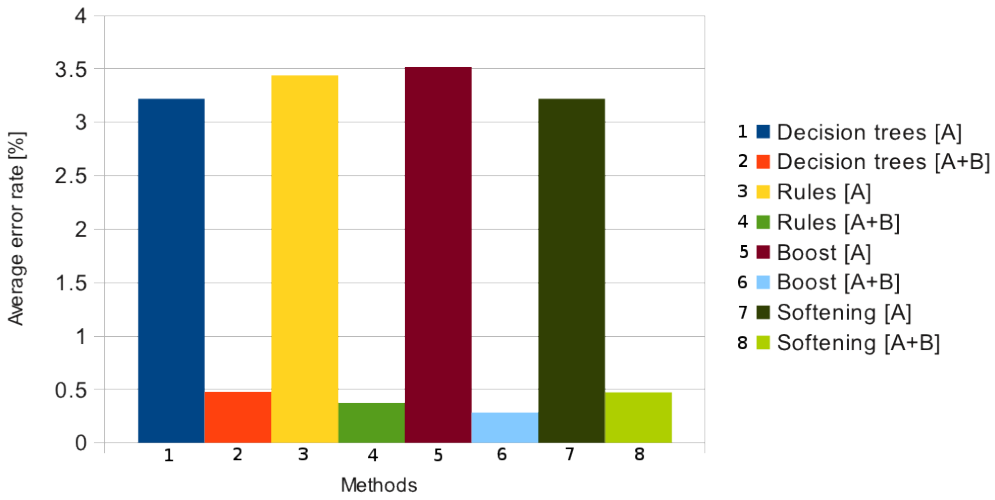

Figure 3: Average Error Rates of the Classifiers [20]

generate the decision tree or rules drastically decreased [26]. In order to test the efficiency of C5.0, we performed a set of tests during which we used various training and classification options. The training statistics were obtained from the data provided by our VBS. During our research, we found relevant the set of arguments and discovered that the best results were obtained using the boosted classifier. The average accuracy fluctuated between 99.3 % and 99.9 %, depending on the number of training and test cases and the amount of data from each case. This behavior is illustrated in Figure 3. It is worth mentioning that in our experiment we considered only 7 different groups of applications and only flows longer than 15 packets. In our small-scale prototype for tests, we decided to limit the number of applications and take into account Skype, FTP, torrent, web traffic, web radio traffic, interactive game traffic, and SSH [20]. The limitation of the flow length was done because we needed to have at least 5 packets to generate the statistics (the first 10 packets of each flow were skipped as their behavior is different than the behavior of the rest of the flow). The detailed description of our methods and results can be found in [20]. The decision tree generated in this step can be used to classify the traffic in the real network.

8 Centralized Monitoring Solution

This paragraph presents the proposed design of the centralized monitoring solution, which can be placed in any point in the network to examine the network QoS.

Because of the heavy load in the high-speed networks, it is not possible to monitor all the flows passing the central point at the same time. Therefore, only the statistics from selected flows can be captured and passed to C5.0. The selection of such flows can be based on two methods: capturing one flow per user and intelligent switching between the flows. From the QoS point of view, it is important to discover the problems with a particular user or to inform the user that the problems experienced by him are the result of problems in the remote network. If it is the user who has the problem, then the problem usually influences all the user's network activity.

Each application has some special requirements regarding the network parameters. When a small congestion occurs, the service level can still be sufficient for P2P file downloads, but Skype communication may be not possible because of big jitter and delays. It is, therefore, not sufficient to monitor one random flow at a time, but we need to monitor a flow which have high quality requirements. Our solution should be built based on the following assumptions:

- Only one flow per user at a time is consistently monitored for QoS.
- Statistics for another random flow per user at a time are passed to C5.0 to discover the application.
- If the application has higher QoS requirements than the currently monitored, switch monitoring to the new flow; if not, stick to the current one.
- If the monitoring of the selected flow discovers problems, start monitoring a few flows at a time to check if this problem lay on the user's side or on the remote side.

Because of the dynamic switching between the flows when determining the application, it is most probable that the system will not be able to capture flows from their beginning. The classifier designed by us, which uses C5.0, is able to determine the application on the basis of the given number of packets from any point in the flow [20].

Monitoring of the QoS can be done in a passive or an active mode. The passive mode relies mostly on time-based statistics, which are obtained directly from the flow passing the measurement point. This way, we can assess the jitter, the burstiness and the transmission speed (both download and upload). Unfortunately, it is not possible to receive the information about the packet loss or the delay for other than TCP streams while using this method. For that reason, additional tools performing active measurements must be involved in the process of estimating the QoS. One option is to use the ping-based approach, as it can measure both the delay and packet loss. Unfortunately, other issues can arise. Ping requests and responses are often blocked by network administrators, or their priority is modified (decreased to save the bandwidth or increased to cheat the users about the quality of the connection). Other options include sending IP packets with various TTL and awaiting *Time Exceeded* ICMP messages, which are

usually allowed to be transmitted in all the networks and their priority is not changed. Active measurements must be done in both directions (from the user and from the remote side). The total packet loss and the delay can be calculated as the sum of the delays and the packet losses from both directions of the flow. Furthermore, the knowledge of the direction that causes the problems can be used to assess if the problems are located in the local network or somewhere outside.

9 Conclusion

This paper shows a novel method for assessing the Quality of Service in computer networks. Our approach involves a group of volunteers from the target network to participate in the initial training of the system, and later in the self-learning process. The accurate data obtained from the volunteers are used by the C5.0 MLA to create the per-application profiles of the network traffic as classification decision trees. The centralized measurement system uses the decision trees to determine the applications associated with the flows passing through the measurement point. This knowledge allows us to precisely define the QoS requirements for each particular flow. To assess the QoS level two methods are proposed: the passive and the active one.

10 Acknowledgment

This research work was conducted in the context of a PhD project in the Section for Networking and Security in the Department of Electronic Systems at Aalborg University. The project is co-financed by the European Regional Development Fund (ERDF) – see http://ec.europa.eu/regional_policy/thefunds/regional/index_en.cfm, and Bredbånd Nord A/S, a regional fiber networks deliverer (see http://www.bredbaandnord.dk/). We are also very grateful, for the possibility to install and test our Volunteer-Based System, to *Gimnazjum nr 3 z Oddziałami Integracyjnymi i Dwujęzycznymi imienia Karola Wojtyły w Mysłowicach*, a high school in Poland.

References

[1] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for Assessing Quality of Service in Broadband Networks. In *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT)*, pages 826–831. IEEE, Phoenix Park, PyeongChang, Korea, February 2012. Accessible: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6174795.

[2] Gerhard Haßlinger. Implications of Traffic Characteristics on Quality of Service in Broadband Multi Service Networks. In *Proceedings of the 30th EUROMICRO*

Conference (EUROMICRO'04), pages 196–204. IEEE, Rennes, France, September 2004. DOI: 10.1109/EURMIC.2004.1333372.

[3] Thomas M. Chen and Lucia Hu. Internet Performance Monitoring. *Proceedings of the IEEE*, 90(9):1592–1603, September 2002. DOI: 10.1109/JPROC.2002.802006.

[4] LIRNEasia Broadband QoSE Benchmarking project, 2008. [Online]. Available: http://lirneasia.net/projects/2008-2010/indicators-continued/broadband-benchmarking-qos-20/.

[5] K. v. M. Naidu, Debmalya Panigrahi, and Rajeev Rastogi. Detecting Anomalies Using End-to-End Path Measurements. In *Proceedings of the 27th Conference on Computer Communications IEEE INFOCOM 2008*, pages 16–20. IEEE, Phoenix, Arizona, USA, April 2008. DOI: 10.1109/INFOCOM.2008.248.

[6] Aboagela Dogman, Reza Saatchi, and Samir Al-Khayatt. An Adaptive Statistical Sampling Technique for Computer Network Traffic. In *Proceedings of the 7th International Symposium on Communication Systems Networks and Digital Signal Processing (CSNDSP 2010)*, pages 479–483. IEEE, Newcastle upon Tyne, England, United Kingdom, July 2010.

[7] Baek-Young Choi, Jaesung Park, and Zhi-Li Zhang. Adaptive Random Sampling for Traffic Load Measurement. In *Proceedings of the IEEE International Conference on Communications (ICC'03)*, volume 3, pages 1552–1556. IEEE, Anchorage, Alaska, USA, May 2003. DOI: 10.1109/ICC.2003.1203863.

[8] Shao Liu, Mung Chiang, Mathias Jourdain, and Jin Li. Congestion Location Detection: Methodology, Algorithm, and Performance. In *Proceedings of the 17th International Workshop on Quality of Service (IWQoS 2009)*, pages 1–9. IEEE, Charleston, South Carolina, USA, July 2009. DOI: 10.1109/IWQoS.2009.5201404.

[9] Jun Li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. Real-time P2P traffic identification. In *Proceedings of the IEEE Global Telecommunications Conference (IEEE GLOBECOM 2008)*, pages 1–5. IEEE, New Orleans, Louisiana, USA, December 2008. DOI: 10.1109/GLOCOM.2008.ECP.475.

[10] Riyad Alshammari and A. Nur Zincir-Heywood. Machine Learning based encrypted traffic classification: identifying SSH and Skype. In *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA 2009)*, pages 1–8. IEEE, Ottawa, Ontario, Canada, July 2009. DOI: 10.1109/CISDA.2009.5356534.

[11] Sven Ubik and Petr Žejdl. Evaluating application-layer classification using a Machine Learning technique over different high speed networks. In *Proceedings of the Fifth International Conference on Systems and Networks Communications*

(ICSNC), pages 387–391. IEEE, Nice, France, August 2010. DOI: 10.1109/ICSNC.2010.66.

[12] Ying Zhang, Hongbo Wang, and Shiduan Cheng. A Method for Real-Time Peer-to-Peer Traffic Classification Based on C4.5. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 1192–1195. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689126.

[13] Jing Cai, Zhibin Zhang, and Xinbo Song. An analysis of UDP traffic classification. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 116–119. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689203.

[14] Riyad Alshammari and A. Nur Zincir-Heywood. Unveiling Skype encrypted tunnels using GP. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, Barcelona, Spain, July 2010. DOI: 10.1109/CEC.2010.5586288.

[15] Kartheepan Balachandran, Jacob Honoré Broberg, Kasper Revsbech, and Jens Myrup Pedersen. Volunteer-Based Distributed Traffic Data Collection System. In *Proceedings of the 12th International Conference on Advanced Communication Technology (ICACT 2010)*, volume 2, pages 1147–1152. IEEE, Phoenix Park, PyeongChang, Korea, February 2010.

[16] Kartheepan Balachandran and Jacob Honoré Broberg. Volunteer-Based Distributed Traffic Data Collection System. Master's thesis, Aalborg University, Department of Electronic Systems, Denmark, June 2010.

[17] Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for classification of traffic in computer networks. In *Proceedings of the 19th Telecommunications Forum TELFOR 2011*, pages 210–213. IEEE, Belgrade, Serbia, November 2011. DOI: 10.1109/TELFOR.2011.6143528.

[18] Tomasz Bujlow, Kartheepan Balachandran, Sara Ligaard Nørgaard Hald, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for research on the Internet traffic. *TELFOR Journal*, 4(1):2–7, September 2012. Accessible: `http://journal.telfor.rs/Published/Vol4No1/Vol4No1.aspx`.

[19] Volunteer-Based System for Research on the Internet, 2012. [Online]. Available: `http://vbsi.sourceforge.net/`.

[20] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for classification of network traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of ICNC'12: 2012 International Conference on Computing, Networking and*

Communications (ICNC): Workshop on Computing, Networking and Communications, pages 244–248. IEEE, Maui, Hawaii, USA, February 2012. DOI: 10.1109/ICCNC.2012.6167418.

[21] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. Classification of HTTP traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of the Fourth IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Web-based Service Architectures (PEDISWESA 2012)*, pages 882–887. IEEE, Cappadocia, Turkey, July 2012. DOI: 10.1109/ISCC.2012.6249413.

[22] Jason But, Philip Branch, and Tung Le. Rapid identification of BitTorrent Traffic. In *Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN)*, pages 536–543. IEEE, Denver, Colorado, USA, October 2010. DOI: 10.1109/LCN.2010.5735770.

[23] Li Jun, Zhang Shunyi, Lu Yanqing, and Zhang Zailong. Internet Traffic Classification Using Machine Learning. In *Proceedings of the Second International Conference on Communications and Networking in China (CHINACOM '07)*, pages 239–243. IEEE, Shanghai, China, August 2007. DOI: 10.1109/CHINACOM.2007.4469372.

[24] Yongli Ma, Zongjue Qian, Guochu Shou, and Yihong Hu. Study of Information Network Traffic Identification Based on C4.5 Algorithm. In *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, pages 1–5. IEEE, Dalian, China, October 2008. DOI: 10.1109/WiCom.2008.2678.

[25] Wei Li and Andrew W. Moore. A Machine Learning Approach for Efficient Traffic Classification. In *Proceedings of the Fifteenth IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'07)*, pages 310–317. IEEE, Istanbul, Turkey, October 2007. DOI: 10.1109/MASCOTS.2007.2.

[26] Is See5/C5.0 Better Than C4.5, 2009. [Online]. Available: `http://www.rulequest.com/see5-comparison.html`.

Tomasz Bujlow is working as a Ph.D. Student in the Section for Networking and Security (NetSec) in the Department of Electronic Systems at Aalborg University in Denmark. He received his Master of Science in Computer Engineering from Silesian University of Technology in Poland in 2008, specializing in Databases, Computer Networks and Computer Systems. Previously, he obtained his Bachelor of Computer Engineering from University of Southern Denmark in 2009, specializing in software engineering and system integration. His research interests include methods for measurement of Quality of Service and traffic classification in computer networks. He is also a Cisco Certified Network Professional (CCNP) since 2010.

Sara Ligaard Nørgaard Hald is working as a Ph.D. student in the Section for Networking and Security (NetSec) in the Department of Electronic Systems at Aalborg University in Denmark. She received her Master of Science in Computer Engineering and Management from Aalborg University in 2002, and has since worked for the Danish Defense and as a consultant specializing in enterprise architecture and cybersecurity. Research interests include threat assessments and attack detection in dedicated networks.

Tahir Riaz is working as an Assistant Professor in the Section for Networking and Security (NetSec) in the Department of Electronic Systems at Aalborg University in Denmark. He received his Master and PhD degrees in Electrical and Electronics Engineering, specializing in Network Planning and Management, from Aalborg University in 2004 and 2008, respectively. He has also worked in Nokia, Linkoping, Sweden. He has authored or co-authored over 70 papers published in conferences and journals. His research interests include access and backbone fiber optic networks, network planning and design, architecture of next generation radio of fiber networks, reliability and QoS issues in large-scale access and core network infrastructures, performance and optimization in networks.

Jens Myrup Pedersen is working as an Associate Professor and the head of the Section for Networking and Security (NetSec) in the Department of Electronic Systems at Aalborg University. His current research interests include network planning, traffic monitoring, and network security. He obtained his Master of Science in Mathematics and Computer Science from Aalborg University in 2002, and his PhD in Electrical Engineering also from Aalborg University in 2005. He is an author/co-author of more than 70 publications in international conferences and

journals, and has participated in Danish, Nordic and European funded research projects. He is also a board member of a number of companies within technology and innovation.

Paper IV

Classification of HTTP Traffic Based on C5.0 Machine Learning Algorithm

Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen

Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark
{tbu, tahir, jens}@es.aau.dk

This paper is reprinted from the *Proceedings of the Fourth IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Web-based Service Architectures (PEDISWESA 2012)*, pages 882–887. IEEE, Cappadocia, Turkey, July 2012.

DOI: 10.1109/ISCC.2012.6249413.

Abstract

Our previous work demonstrated the possibility of distinguishing several groups of traffic with accuracy of over 99 %. Today, most of the traffic is generated by web browsers, which provide different kinds of services based on the HTTP protocol: web browsing, file downloads, audio and voice streaming through third-party plugins, etc. This paper suggests and evaluates two approaches to distinguish

various types of HTTP traffic based on the content: distributed among volunteers' machines and centralized running in the core of the network. We also assess the accuracy of the centralized classifier for both the HTTP traffic and mixed HTTP/non-HTTP traffic. In the latter case, we achieved the accuracy of 94 %. Finally, we provide graphical characteristics of different kinds of HTTP traffic.

Keywords

traffic classification, computer networks, HTTP traffic, browser traffic, C5.0, Machine Learning Algorithms (MLAs), performance monitoring

1 Introduction

The assessment of the Quality of Service (QoS) in computer networks is a challenging task because different kinds of traffic flows (voice and video streaming, file download, web browsing) have different requirements. Therefore, to estimate the performance, we need to know what type of data flow is currently being assessed. There are many methods for distinguishing computer network traffic, including the classification by ports, Deep Packet Inspection (DPI), or statistical classification [1]. We compared them in [2] and we assessed that these methods are not sufficient for the real-time identification of HTTP traffic.

We had two possible approaches to classify the flows in a high-speed computer network infrastructure: centralized and distributed. We implemented the distributed approach as the Volunteer-Based System (VBS) and presented in [2]. VBS clients installed on users' computers collect the data together with the name of the corresponding application. The necessary statistical parameters are calculated on the client side and sent to the database server. We designed the centralized solution as a flow-examining-application installed in any point of the network. All the flows passing through that point are captured and assigned to a particular application class by the C5.0 Machine Learning Algorithm (MLA) [3]. As training data, we used the data collected by VBS. The proposed design of a solution for estimating QoS using both these approaches and combining passive and active measurements was described in [4]. The accuracy of the distributed QoS assessment solution is approaching 100 %, as it uses the process names taken directly from the system sockets during the classification. The accuracy of our centralized QoS assessment was assessed to be 99.3–99.9 %, due to the C5.0 classification error estimated based on our previous approach to classify 7 different applications [3].

In previous papers, we assumed that one application carries only one type of traffic and, for this reason, we took into account only applications fulfilling this criterion. However, the data collected by VBS showed that nowadays majority of traffic is generated by HTTP-based applications as web browsers. Until now, we were treating this kind of

traffic as a general *web* traffic class, which in effect consisted of interactive traffic (web pages), audio and video streams, and big file downloads (including big video files downloaded directly from a website by the user or downloaded indirectly by a web player, as YouTube). Flows carrying different kinds of content can have different characteristics and QoS requirements [5] and, therefore, they need to be distinguished and processed in different ways. The measured characteristics of different content types found within HTTP flows are shown at the end of this paper. All these factors lead to the conclusion that during QoS assessment, we are interested in the type of the traffic (taking into account both the type of the content and the type of the content delivery, as streaming or casual download), not in the application which it generates. In this paper, we present and evaluate a method for recognizing different kinds of HTTP traffic.

Other methods for the classification of HTTP traffic are shown in [6] and [7]. In [6], the authors propose to use the size of the flow and the number of flows associated with the same IP address to determine the character of the traffic by 3 different MLAs. Unfortunately, this approach requires to have the traffic collected in advance, and in consequence, it is not suitable for the real-time classification needed for QoS assessment purposes. The method described in [7] is based on keyword matching, flow statistics and a self-developed algorithm. This approach also does not fulfill our needs because it requires processing entire flows: first to match the signature, then to extract statistics, such as the number of packets contained by the flow. As opposite, our centralized solution is able to classify the data based on 35 consecutive packets from a random point of a flow. As a consequence, we can monitor flows very quickly.

The remainder of this paper gives the overview of our solutions for the distributed and centralized classification of network traffic and our methods for providing precise input data, describes the results, and shows various traffic profiles. We assessed the accuracy of the classification while changing parameters in the algorithm. The data used in our experiments originate from 5 private machines running in Denmark and in Poland as well as 18 machines installed in computer classrooms in *Gimnazjum nr 3 z Oddziałami Integracyjnymi i Dwujęzycznymi imienia Karola Wojtyły w Mysłowicach*, a high school in Poland.

2 Centralized Classification Method

We designed the centralized method to be used in the core of the network. 35-packet long snippets from the selected flows are inspected by the statistics generator, which calculates the values of the relevant parameters. Based on the calculated statistics, the C5.0 MLA is able to predict the traffic class of the flow. The first and the most important issue in our solution was how to train the classifier properly. As a consequence, we designed and implemented an algorithm, which uses pre-classified browser traffic to generate training cases for different classes of traffic. The description of the algorithm is shown in Figure 1.

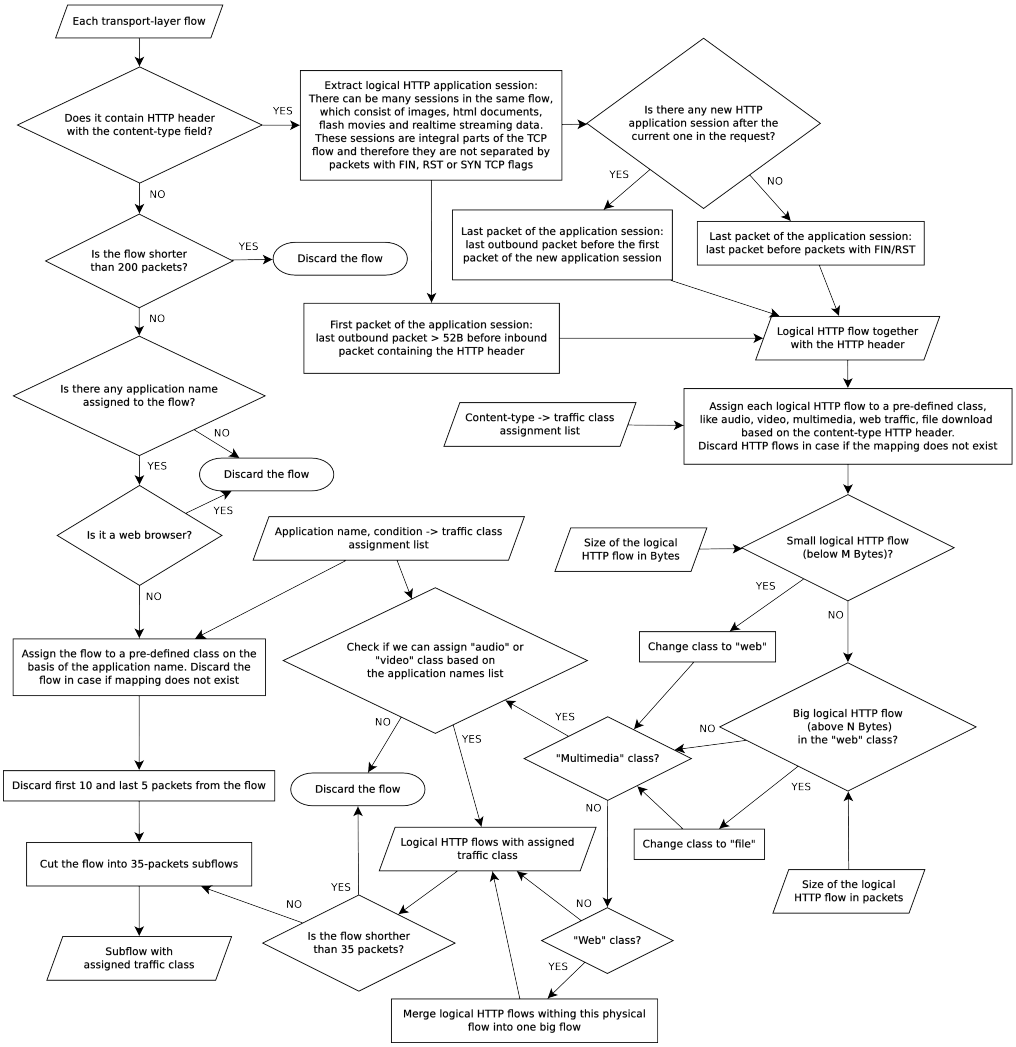

Figure 1: Overview of the Method for Obtaining the Training Data

Browser traffic can be classified based on two different approaches: by using HTTP headers, or application names and additional flow conditions like ports. Table 1 contains examples of different services accessible by Firefox web browser, together with the information about the chosen method of classification. As shown, the type of the content delivered by most services can be classified accurately by the *content-type* field in the HTTP header. Unfortunately, in some cases, we are not able to distinguish HTTP audio from HTTP video streams (as shown in the case of *application/x-mms-framed* content type, used both for streamed audio and video content). However, streamed multimedia content is often played by plugins which use the Real-Time Messaging Protocol (RTMP) instead of HTTP, so the content can be separated using plugin names (such as *plugin-container*) and RTMP remote port (1935). From the QoS point of view, the problem is that based on the *content-type* field, we cannot distinguish streamed multimedia content from multimedia files embedded on websites (such as YouTube) and just downloaded in the background to the user's computer, because they can use identical values of the *content-type* field, for example, *audio/mpeg*. The same situation happens when a user downloads a video or audio file explicitly by using a download link. Therefore, for the purpose of this experiment, we placed all the flows delivering the *video* content to the same class, regardless, if the content was streamed or downloaded.

As the first step, we need to decide if we are dealing with an HTTP-based flow or another kind of transport-layer flow. For this purpose, we examine each packet in the flow and check if the HTTP header exists. If yes, we look for the *content-type* field. If we can obtain the information, the preferred way of processing is always to handle the flow as an HTTP-based flow, as it allows to recognize different kinds of flows generated by one application. Short flows (below 200 packets in the case of regular flows, and below 35 packets in the case of HTTP-based flows) are discarded because they seem to be less useful from the QoS measurement point of view.

Table 1: Examples of Different Services Provided by Web Browsers

Media	Location	Application	Content	Classification
Internet radio				
The Voice	http://www.thevoice.dk/popup/popup.php?tab=radio	firefox	audio/mpeg	By content type
NOVAfm	http://www.novafm.dk/popup/popup.php?tab=radio	firefox	audio/mpeg	By content type
Radio 3	http://www.radio3.dk/sites/all/modules/netplayer/player.php	plugin-container		Impossible
RMF FM	http://www.rmfon.pl/play,5	plugin-container	audio/aacp	By content type
ESKA	http://www.eska.pl/player?streamId=101	firefox	audio/mpeg	By content type
CNN radio	http://radioradio7.com/radio/CNN.html	totem-plugin-	application/x-mms-framed	By content type
Embedded audio				
Wrzuta.pl	http://www.wrzuta.pl	firefox	audio/mpeg	By content type
Video on Demand				
Youtube	http://www.youtube.com/	firefox	video/x-flv	By content type
Ipla	http://www.ipla.pl	iplalite	video/x-flv	By content type
Onet Video	http://www.onet.pl	firefox	video/x-flv	By content type
CNN Video	http://edition.cnn.com/video/	firefox	video/x-flv	By content type
Wrzuta.pl	http://www.wrzuta.pl	firefox	video/mp4	By content type
Internet TV				
Justin.tv	http://www.justin.tv/	plugin-container		By app name and remote port 1935
Al-Jazeera	http://www.aljazeera.com/watch_now/	plugin-container		By app name and remote port 1935
PDR	http://www.pdr.pl	totem-plugin-	application/x-mms-framed	By content type
File download				
File 1	http://download.oracle.com/otn-pub/java/jdk/7u1-b08/jdk-7u1-solaris-sparc.tar.Z	firefox	application/x-compress	By content type
File 2	http://www.skatnet.dk/test/testfile.avi	firefox	video/x-msvideo	By content type as *video*. However, the most proper class would be *file download*

2.1 Regular Transport-Layer Flows

Regular flows are processed based on the assignments between the application names and traffic classes. Most applications are specialized to handle specific types of traffic (voice conversations for Skype, file transfer for FTP clients, or interactive traffic for games), but they also generate background traffic. For example, Skype shares a distributed users' directory, free file transfer clients tend to download advertisements to display them on the screen when doing their job, and games have control connections to the main server. These flows, acting as noise, are usually quite short. To eliminate their impact, we decided to discard all flows shorter than 200 packets. If there is no application name assigned to the flow, the flow is discarded. Flows associated with HTTP-based applications (like web browsers) are discarded as well, because they are not recognized as HTTP flows and their type is unknown. Then the flows are checked against the assignments between the application names and traffic classes. If we cannot find any match, the flow is discarded as well. As it is written in [2], around the first 10 and the last 5 packets of each flow have different characteristics of size parameters than the other packets. As a result, these packets are cut out of the flow. Next, the flow is split into 35-packet subflows, which are provided to the statistics generator. The generated statistics are given as the input to the C5.0 classifier as training or test data. It was shown in [2] that further increasing of the number of packets in the subflow does not improve significantly the accuracy of the classifier. Using the reasonably smallest number of flows allows to perform faster traffic classification and saves system resources, what allows to process more flows at a time.

2.2 HTTP-Based Transport-Layer Flows

Dealing with HTTP-based flows is more complex, as one transport-layer flow by HTTP can carry multiple different files as text, images, audio, and video. For this reason, we split the transport-layer flow into entities carrying different files or streamed content (called later *separate HTTP flows, as they have separate HTTP headers*), which are mapped to a traffic class based on the *content-type* field in the HTTP header. We found that the *content-type* field in the HTTP header is present in and only in the first inbound packet of a new logical HTTP flow. If the mapping does not exist, the HTTP flow is discarded.

We decided to specify the following traffic classes: *audio*, *file download*, *multimedia*, *video*, and *web*. The *multimedia* class was assigned to traffic with content-types, which could carry audio as well as video (regardless if the content was streamed or downloaded, as based on the content it is impossible to detect). The *file download* class was assigned to big file downloads (however, except the multimedia files, which were assigned to one of the multimedia classes), and the *web* class to the traffic produced by interactive web browsing. It was very hard to define what the interactive web browsing is, but we decided to create this class as characteristics of transport-layer flows carrying multiple

small files like HTML documents, web images and stylesheets are different than the characteristics of downloads of big files. It can be even more complicated because small few-second videos and animations on websites behave more like the interactive traffic than the real video traffic. On the other hand, big images embedded on websites behave like file downloads. Therefore, we decided to consider all the small (below M Bytes) HTTP flows as web interactive traffic, and all the big *web* flows (above N Bytes) as *file download* traffic. Values for M and N are intended to be chosen experimentally. All the interactive HTTP *web* flows within the transport-layer flow were merged together into one big *web* flow. It makes no sense to calculate statistics for each small web element (HTML file, images, etc) separately, because they are visible across the network as one interactive flow, so they must be processed holistically to assure the proper assessment of the QoS level.

The entities carrying the *multimedia* content must be re-classified as either *audio* or *video*, because they have different characteristics and requirements. We use the assignments between the application names and traffic classes to see if the application assigned to that flow is purely audio or video oriented. If not, the flow is discarded. Flows shorter than 35 packets are dropped to ensure compatibility with processing regular transport-layer flows. Finally, the flow is split into 35-packets subflows, which are provided to the statistics generator. The generated statistics are the input given to the C5.0 classifier as the training or the test data.

3 Data Sources

The algorithm of obtaining the training data uses three external sources: a set of transport-layer flows, the assignments between the application name and traffic classes, and the assignments between the content types and traffic classes. The origin of the mentioned data sources is described below.

3.1 Transport-Layer Flows

The transport-layer flows are obtained by our Volunteer-Based System (VBS), whose architecture and implementation was described in [2]. We implemented several major changes to the system since it was published in order to make it capable to process the information about HTTP content types.

First, we used the JNetPcap library to detect the HTTP header in each packet of the flow and, in case of presence, to extract the values of the *content-type* field. The field is always present in the first incoming packet of the logical HTTP flow inside the transport-layer flow. Obtaining this information allows us not only to detect the class of the traffic but also to separate logical HTTP flows within one transport-layer flow. The extracted values of the *content-type* field are associated with particular packets and sent to the server where they are stored in a separate table and associated with the corresponding

Table 2: Mapping Application Names to Traffic Classes

Name	Requirement	Class
amule		p2p
dropbox		file download
filezilla		file download
http		file download
java		file download
libgcflashplay	remote_port = 1935	video
plugin-container	remote_port = 1935	video
skype	protocol_name = 'UDP'	audio
ssh		ssh
steam		file download
utorrent		p2p
wget		file download

packets. This way, we are able to keep track of the content types in one place and save space. Due to obtaining all the relevant information directly from the client machines, VBS fulfills two roles. First, it is an independent distributed solution able to classify network traffic in real-time from the machine where it is installed. Second, it delivers the data for training the C5.0, which is used in the centralized classification approach. We must admit that turning on the HTTP header inspection did not increase the CPU usage in a measurable way, so there was no need to implement any optimization methods (like processing only a part of the flows, inspecting only a a part of the packets in a flow, and so on).

3.2 Assignments between the Application Names and Traffic Classes

Obtaining the mappings between application names and traffic classes is quite straightforward and it was done in the way described below. The results for our case are shown in Table 2. It is worth mentioning that one element can possibly match multiple classes. For example, a *p2p* flow can carry a file (so it can be in fact a good match for the *file download* class). The *audio* and *video* classes carry in this case only the streamed content. The *http* process in a standard Ubuntu application, which is responsible for downloading files for system purposes, as system upgrades.

- Extract all the application names from the flows, which contain at least 5000 packets in total. This limitation was made to prevent including in the listing applications which generated only small amounts of data, because they are not sufficiently representative.
- Change all the names to lowercase and trim the whitespace from both ends. Then write the list to a Comma Separated Values (CSV) file.

Table 3: Mapping Content Types to Traffic Classes

Class	Content Type
audio	audio/aac, audio/aacp, audio/mpeg, audio/x-mpegurl, audio/x-pn-realaudio-plugin, audio/x-scpls
file download	application/binary, application/force-download, application/octet-stream, application/octetstream, application/pdf, application/rar, application/x-bzip2, application/x-compress, application/x-debian-package, application/x-gzip, application/x-msdos-program, application/x-msdownload, application/x-redhat-package-manager, application/x-tar, application/x-xpinstall, application/x-zip-compressed, application/zip, binary/octet-stream
multimedia	application/x-mms-framed, application/ogg
video	application/x-fcs, flv-application/octet-stream, video/mp4, video/ogg, video/webm, video/x-flv, video/x-m4v, video/x-ms-asf, video/x-msvideo
web	application/atom+xml, application/gif, application/java-archive, application/javascript, application/js, application/json, application/ocsp-request, application/ocsp-response, application/opensearchdescription+xml, application/pkix-crl, application/rdf+xml, application/rss+xml, application/smil, application/soap+xml, application/x-amf, application/x-director, application/x-font, application/x-httpd-cgi, application/x-java, application/x-java-archive, application/x-javasc, application/x-javascri, application/x-javascrip, application/x-javascript, application/x-ns-proxy-autoconfig, application/x-pkcs7-crl, application/x-sdch-dictionary, application/x-shockwave-flash, application/x-silverlight-app, application/x-woff, application/x-ww, application/x-www, application/x-x509-ca-cert, application/xaml+xml, application/xhtml+xml, application/xml, banner/jpg, font/woff, httpd/unix-directory, image/bmp, image/gif, image/ico, image/jpeg, image/jpg, image/pjpeg, image/png, image/svg+xml, image/vnd.microsoft.icon, image/x-ico, image/x-icon, image/x-ms-bmp, image/x-png, multipart/byteranges, multipart/form-data, text/css, text/html, text/javascript, text/json, text/plain, text/vdf, text/x-c, text/x-cross-domain-policy, text/x-gwt-rpc, text/x-javascript, text/x-js, text/x-json, text/x-perl, text/xml

- Manually assign a traffic class to all the rows in the file. Add a condition as a part of an SQL statement, if needed (for example by restricting the transport layer protocol to UDP or including only flows matching particular port numbers). Some of the applications also can generate background traffic, which must be cut off (like Skype, which beside the main voice UDP flow generates numerous TCP connections to other clients to exchange the distributed users' directory). If the application is unknown or it can handle different kinds of traffic which cannot be separated by a SQL statement, it should be deleted from the list.

3.3 Assignments between the Content Types and Traffic Classes

To be able to map the logical HTTP flows to a traffic class, we needed to create a mapping table based on the information contained in the database. This resulted in the assignments shown in Table 3. The process of obtaining these mappings is described

	N=0	N=100	N=200	N=300	N=500	N=1000
M=0	17.9					
M=30		17.3	17.2	17.2	17.2	17.2
M=60		17.1	17.2	17.2	17.2	17.2
M=100		17.1	17.3	17.2	17.2	17.3
M=150			17.3	17.3	17.3	17.3
M=300				17.2	17.2	17.2
M=500					17.3	17.2

Figure 2: Classification Error Rate [%] for Different Values of M and N

below. It is worth mentioning that one element can possibly match multiple classes, as the classes present various levels: content (as *audio*) or behavior (as *file download*). For example, an *audio* flow can carry a streamed content from a web radio or an audio file (so it can be in fact a good match for the *file download* class).

- Extract all the values of the content types from the packets, changing their names to lowercase and trimming the whitespace from both ends.
- Remove from the content type everything beyond the type itself, for example, the information about the used encoding. Then, write the list to a CSV file.
- Manually assign a traffic class to all the rows in the file. If the content type cannot be verified, delete the row. If the content-type can correspond both to the audio and to the video traffic, assign the *multimedia* class.

Unfortunately, relying on mappings between the traffic classes and the content types is not always accurate and consistent regarding the QoS assessment. For example, a movie downloaded (directly by the user from a website or indirectly in the background by the browser from YouTube) is marked as the *video* flow, as it carries video content. However, regarding the QoS requirements, the *video* class should contain only streamed video content, and the most appropriate action in this case is to mark the flow as the *file download*, but there is no simple way to obtain knowledge about the purpose of the traffic beside asking the user what he is currently doing.

4 Classification by C5.0

During the experiment, we used the same sets of classification attributes (A plus B) as used in [3]. We performed a normal decision-tree based classification for two cases: only for HTTP traffic, and for the mixed HTTP/non-HTTP traffic. In the first case, we tried to estimate the optimal values for the parameters M and N in the algorithm, so we made several tries with various values of M and N while observing how it affects the classification error. Both parameters equal to 0 mean that the mechanism of switching

Table 4: Misclassification Matrix [%] for the HTTP Traffic

Class / Classified	as *audio*	as *file download*	as *video*	as *web*
audio	98.04	0.40	1.03	0.53
file download	0.00	78.08	20.96	0.95
video	0.02	12.67	85.95	1.37
web	0.26	10.67	9.52	79.56

Table 5: Misclassification Matrix [%] for the Mixed HTTP/Non-HTTP Traffic

Class / Classified	as *audio*	as *file download*	as *p2p*	as *ssh*	as *video*	as *web*
audio	95.89	0.64	1.37	0.00	1.72	0.38
file download	0.01	86.03	0.72	0.00	12.51	0.73
p2p	0.00	0.13	99.85	0.00	0.01	0.00
ssh	0.00	0.75	0.00	96.79	0.35	2.10
video	0.02	12.85	0.06	0.00	86.12	0.95
web	0.13	11.88	0.16	0.02	9.40	78.42

flows between the *web* and the *file download* classes is turned off. The matrix of the classification error for HTTP flows while using different values of M and N is shown in Figure 2.

As shown, the accuracy of the classifier is independent of the lower and upper limits (M and N) for the interactive web traffic. Moreover, we can turn off the changing-class mechanism without significant decrease of the accuracy. We observe this behavior, because in order to test the classification accuracy, we use a disjoint set of data obtained in the same way as the set used for the training purposes, so it uses the same values of M and N. It means that the proper values for M and N should be estimated through observations of the traffic. It ensures that the *web* class contains as much of the real interactive browser traffic as possible, but as the least of the multimedia and the file transfer traffic.

For the HTTP traffic (the misclassification matrix is shown in Table 4 (M=100, N=300)), we have two major observations. First, we are able to distinguish the *audio* and the interactive *web* traffic among different kinds of HTTP traffic. In our case, the *audio* group contained mostly web radios, which are of the streaming characteristic. Contrary to that, the *video* traffic and the regular *file download* transfers are often confused between themselves, as in fact, our *video* group contained in significant majority video files downloaded by a web browser, so their packet-level characteristic is the same as other file downloads. The average error rate was in this case 17.0 %.

During the second part of the experiment, we used full data sets containing both the HTTP and the non-HTTP traffic (the misclassification matrix is shown in Table 5). For the non-HTTP traffic, we specified the following traffic classes: *audio*, *file download*, *p2p*, *ssh* and *video*. The non-HTTP video transfers were mostly streams played mostly through third-party plugins in the browser, such as Adobe Flash. The average error rate was in this case 6.0 %. The number of the misclassifications between the *file download*

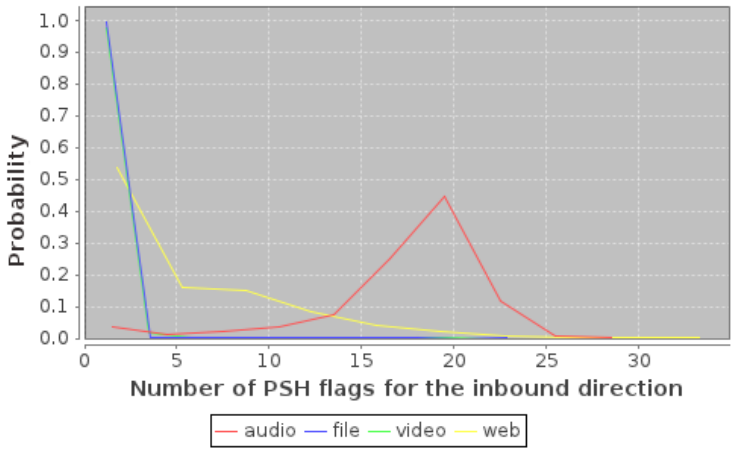

Figure 3: Distribution of Number of PSH Flags for the Inbound Direction

and the *video* classes decreased, as this time the *video* class contained more elements of the streaming characteristic than in the previous case.

5 Traffic Profiles

Based on the output from C5.0, we found the most used classification attributes to distinguish different types of the HTTP traffic. We chose two of them (the number of PSH flags for the inbound direction and the total payload size) to perform the graphical analysis. The distributions of these attributes shown in Figure 3 and in Figure 4 confirm that the *audio* and the *web* traffic differ significantly between each other, and from the *video* traffic and the big *file download* transfers. The number of the PSH flags increases when the content needs to be delivered to the client without delays. It proves that we can easily catch HTTP-based audio traffic, which is the most fragile for network performance issues. It justifies a need for the separate group of interactive *web* traffic as well. During this experiment, we used M=100 and N=300 in the algorithm generating the cases.

6 Conclusion

This paper presents two novel methods for content-based recognizing different kinds of HTTP traffic in computer networks. The distributed method implemented among VBS clients uses the *content-type* fields in the HTTP headers to extract logical HTTP flows from the transport-layer flows. Later, the traffic classes are assigned based on the

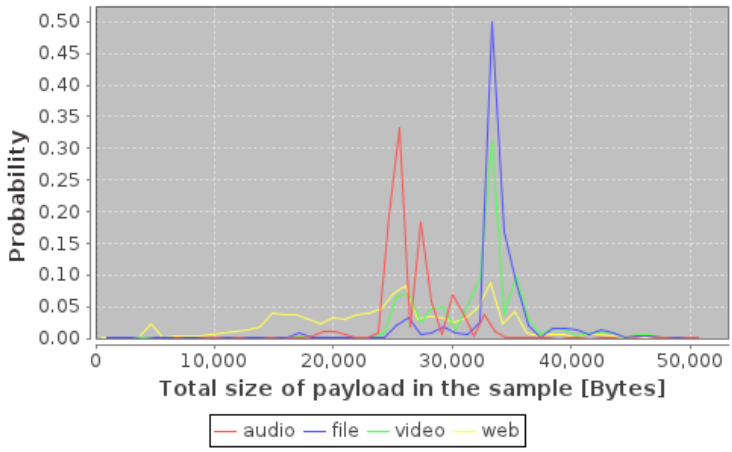

Figure 4: Distribution of Total Payload Size in the Sample

particular types of the content. The centralized method is able to distinguish different content types transported by HTTP in the central point of the network based on the C5.0 MLA. We have shown that the MLA-based classifiers are not able to distinguish different types of the content transported by HTTP when the other flow characteristics (beside the content itself) are the same. The inability to distinguish between the video files and other binary files transported by HTTP caused the high average classification error rate (17.0 %). However, we demonstrated that the classifier did not have problems with recognizing interactive voice traffic, as it was originated mostly by streamed web radios. The last step of our experiment was to classify the mixed HTTP/non-HTTP traffic. In this case, we achieved much lower error rate of 6.0 %, as we included non-HTTP video streams from online TVs.

References

[1] Jun Li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. Real-time P2P traffic identification. In *Proceedings of the IEEE Global Telecommunications Conference (IEEE GLOBECOM 2008)*, pages 1–5. IEEE, New Orleans, Louisiana, USA, December 2008. DOI: 10.1109/GLOCOM.2008.ECP.475.

[2] Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for classification of traffic in computer networks. In *Proceedings of the 19th Telecommunications Forum TELFOR 2011*, pages 210–213. IEEE, Belgrade, Serbia, November 2011. DOI: 10.1109/TELFOR.2011.6143528.

[3] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for classification of network traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of ICNC'12: 2012 International Conference on Computing, Networking and Communications (ICNC): Workshop on Computing, Networking and Communications*, pages 244–248. IEEE, Maui, Hawaii, USA, February 2012. DOI: 10.1109/ICCNC.2012.6167418.

[4] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for Assessing Quality of Service in Broadband Networks. In *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT)*, pages 826–831. IEEE, Phoenix Park, PyeongChang, Korea, February 2012. Accessible: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6174795.

[5] Gerhard Haßlinger. Implications of Traffic Characteristics on Quality of Service in Broadband Multi Service Networks. In *Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)*, pages 196–204. IEEE, Rennes, France, September 2004. DOI: 10.1109/EURMIC.2004.1333372.

[6] Kei Takeshita, Takeshi Kurosawa, Masayuki Tsujino, Motoi Iwashita, Masatsugu Ichino, and Naohisa Komatsu. Evaluation of HTTP video classification method using flow group information. In *Proceedings of the 14th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)*, pages 1–6. IEEE, Warsaw, Poland, September 2010. DOI: 10.1109/NETWKS.2010.5624929.

[7] Samruay Kaoprakhon and Vasaka Visoottiviseth. Classification of audio and video traffic over HTTP protocol. In *Proceedings of the 9th International Symposium on Communications and Information Technology (ISCIT 2009)*, pages 1534–1539. IEEE, Icheon, Korea, September 2009. DOI: 10.1109/ISCIT.2009.5341026.

Paper V

Obtaining Internet Flow Statistics by Volunteer-Based System

Jens Myrup Pedersen and Tomasz Bujlow

Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark
{jens, tbu}@es.aau.dk

This paper is reprinted from the *Fourth International Conference on Image Processing & Communications (IP&C 2012), Image Processing & Communications Challenges 4, AISC 184*, pages 261–268. Springer Berlin Heidelberg, Bydgoszcz, Poland, September 2012.
DOI: 10.1007/978-3-642-32384-3_32.

Abstract

In this paper we demonstrate how the Volunteer Based System for Research on the Internet, developed at Aalborg University, can be used for creating statistics of Internet usage. Since the data are collected on individual machines, the statistics can be made on the basis of both individual users and groups of users, and as such be useful also for segmentation of the users into groups. We present results with

data collected from real users over several months; in particular we demonstrate how the system can be used for studying flow characteristics - the number of TCP and UDP flows, average flow lengths, and average flow durations. The paper is concluded with a discussion on what further statistics can be made, and the further development of the system.

1 Introduction

Understanding the behavior of Internet traffic is crucial in order to model traffic correctly, and to create realistic scenarios of future Internet usage. In particular, understanding the behavior of different kinds of traffic makes it possible to create scenarios of increasing/decreasing particular amounts of traffic. The models can then be used for the analysis and/or simulations of distribution and backbone networks under different scenarios. The application of different provisioning and traffic engineering techniques can be tested as well.

Traffic statistics are today often made by Internet Service Providers (ISPs), who monitor the activity in their networks. However, often ISPs consider these data to be private and are not keen on sharing with researchers. Some traces are publicly available, such as the Caida data sets [1]. Even with access to traces from ISPs or other, traffic monitored in the network cores suffers from missing important statistics that can only be known accurately at the sources - such as inter-arrival times between packets and flow durations. It should be noted that the literature covers a number of interesting studies, where researchers have gained access to real data [2, 3]. In the latter, the data are collected at the broadband access router, which is quite close to the generating source.

There are also a large number of commercial tools for monitoring traffic in Local Area Networks (e.g. on Internet gateways), such as Cisco Netflow [4]. These can provide useful insights to traffic, but without collecting traffic from many different networks it does not give a good overview of how the Internet traffic looks like.

The open-source Volunteer Based System (VBS) for Research on the Internet, developed at Aalborg University, seeks to avoid these problems by collecting the traffic from a large number of volunteers, which are agreeing to have their Internet usage monitored and statistics collected for research purposes. This provides statistics from the point where the traffic is generated, meaning that quite precise statistical data can be obtained. Moreover, it is also possible to monitor which applications are opening the sockets, and thus get the precise picture of the behavior of different applications. The general idea was first described in [5] and a preliminary limited prototype was implemented in [6]. The current system design was announced in [7], while more technical details on later refinements can be found in [8]. Other papers ([9–11]) demonstrate various applications of our system.

In this paper, we show how the system can be used for generating statistics at the

flow level. The paper is organized as follows: First, in Section 2, we describe how the data collection is made and how the statistics are extracted. In Section 3, we present the results, and in Section 4, we conclude the paper and discuss the further work.

The authors would like to stress that the system is based on open source software, published on SourceForge [12]. We would like to take the opportunity to encourage other researchers to use the system for collecting Internet traffic information, and to the widest possible extend share the data traces with the research community.

2 Data Collection and Extracting Statistics

In this section, we briefly describe the fundamentals of VBS, with a particular focus on the parts that influence the monitoring, data collection, and extraction of statistics. For more details, please refer to our previous paper [8].

For each volunteer, the system monitors both ingoing and outgoing traffic on his/her computer. Storing all these data, and transferring them back to the server, would be a huge task, if not impossible given the limited upload capacity available on many standard Internet connections. Therefore, the data are saved as follows, and transmitted to our central server as:

- For each flow, information is stored about e.g. source and destination IP addresses and port numbers, flow start time, flow end time, number of packets, protocol (TCP or UDP) as well as the flow ID. Moreover, the information about the process name, which has opened the socket is collected. This feature provides valuable information allowing us to characterize the traffic created by different applications.
- For each packet, the main information is the packet size and the relative time stamp. Moreover flags from the header are stored, as well as the information about the packet direction and flow ID.

In this way, all relevant information is stored, while the requirements in the terms of memory and network usage are kept at a minimum. Also, no payload is stored at any time, which is an advantage with respect to privacy and security. One privacy concern has been the transfer of source and destination IP addresses. In the current implementation, the IP addresses are hashed before being transferred to the server. However, since the hash function is known (open source), and since the number of IP addresses in IPv4 is limited, it is not difficult to determine the original IP address.

The purpose of this particular study was to demonstrate the usage, so focus was on obtaining and presenting data from a limited number of users prior to run more large-scale experiments. The statistics were obtained from 4 users during the period from January to May 2012. One of the four users (User 4) did not join the system until late April, and thus only participated for the last 2-3 weeks of the study. Due to being a heavy user, the amount of data collected from this machine is higher than from any

Table 1: The Numbers of TCP and UDP Flows for All Users as Well as for the Individual Users. The Number in Parenthesis Shows the Distribution

User	Number of UDP Flows	Number of TCP Flows
All	4770315 (55%)	386530 (45%)
1	446692 (35%)	820927 (65%)
2	3142581 (60%)	2084590 (40%)
3	693389 (52%)	642740 (48%)
4	487653 (60%)	315273 (40%)

of the other participants, despite the shorter participation. During the time of study, all the traffic from the users were collected by the system as described above, and the data stored into our central database.

The four users can be described as follows:

- User 1 - Private user in Denmark
- User 2 - Private user in Poland
- User 3 - Private user in Poland
- User 4 - Private user in Denmark

With the data collected, a wide variety of studies can be conducted. For this paper, we chose to analyze only the flow data (not the packet data), since the amount of data makes it more manageable. As the main purpose is to demonstrate the usefulness of the system, we chose to derive the following statistics:

- Amount of TCP and UDP flows
- Average flow lengths for TCP and UDP flows
- Average flow durations for TCP and UDP flows
- Top 5 applications (measured on the number of flows)

The statistics are done for the individual users as well as for the users altogether.

3 Results

3.1 TCP and UDP Flows

The distributions of TCP and UDP flows are shown in Table 1. It can be seen that both the number of flows and the distribution between TCP and UDP vary quite a bit between the different users.

Table 2: Average Flow Lengths for TCP and UDP Flows

User	Average Length of UDP Flows [packets]	Average Length of TCP Flows [packets]
All	72	81
1	5	110
2	90	80
3	34	29
4	72	114

Table 3: Average Flow Durations for TCP and UDP Flows

User	Average Duration of UDP Flows [seconds]	Average Duration of TCP Flows [seconds]
All	33	26
1	1	32
2	41	25
3	28	7
4	19	55

3.2 Flow Lengths and Durations

The distributions of flow lengths (the number of packets per flow) for TCP and UDP for the different users are shown in Table 2. It is quite interesting to observe that the flow lengths for both TCP and UDP are so different between the different users, indicating a different Internet usage. It should be noted that with the data in the system, it is possible to make a more detailed analysis of the distribution of flow lengths, not only for the different users but also for each application used by each user.

The distribution of flow durations (in seconds) is shown in Table 3. It seems that for users 1-3 the users who have longer average flows also have longer average flow durations. However, user 4 seems to have quite short flow durations even though the flows are quite long. Even though a more thorough analysis is required to explain this in detail, we assume it is due to the user being on a fast Internet connection. However, the type of traffic with generally longer flows probably also plays a role.

3.3 Top 5 Applications

Analyzing the applications is more challenging than deriving the other parameters. First, we did not manage to collect the socket names for a substantial number of the flows. This is mainly concerning very short flows, where the opening time of the socket is so short that it is not captured by the socket monitor. Secondly, what we obtain in order to identify an application is really the process name. For this study, 240 different process names were identified. Further work is needed in order to group these into applications, and for this study, we just list the top 5 process names. It should be noted that it is not a trivial task to determine how e.g. browser plugins should be grouped and

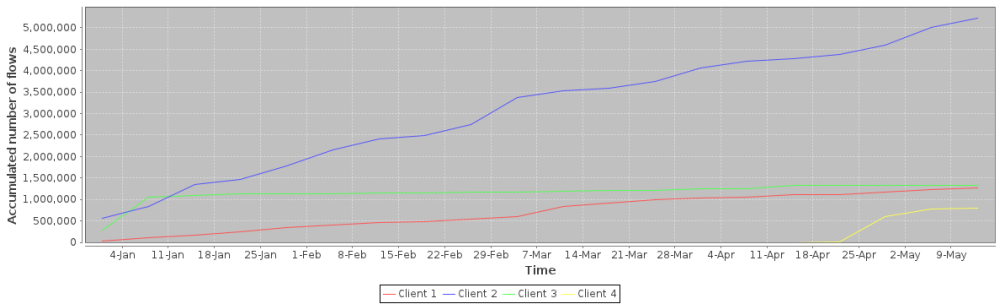

Figure 1: Cumulative Number of Flows for All Users

categorized. The top application names for the different users are shown in Tables 4–8.

Based on the information obtained by the system, it is possible to make additional statistics, taking e.g. the flow lengths of different applications into account. Also packet statistics (e.g. packet lengths) can be taken into account, providing a quite precise picture of what applications are taking up most bandwidth for the different users.

Without going into a more detailed data analysis, we did an observation regarding the unknown flows, which is worth highlighting. The *unknown* flows account for a large number of the total flows. However, the flows have the average length of 2 seconds and the average of 11 packets, indicating that it is not such a large share of the total traffic. These *unknown* flows are almost equally shared between TCP(53%) and UDP(47%).

3.4 Cumulative Number of Flows

The distribution of the cumulative number of flows for the 4 users during the time of our experiment is shown in Figure 1.

4 Conclusion and Discussion

In this paper, we have demonstrated how the Volunteer Based System for Research on the Internet developed at Aalborg University can be used for creating statistics of Internet traffic, specifically within the studies of flows and their properties.

Future research will focus on developing efficient methods for extracting relevant information from the packet statistics. This can provide even more valuable information about the flows, for example, on average packet sizes of different flows (and the distribution of packet sizes), inter-arrival times between packets, and the number of successful vs. unsuccessful connections for different kinds of traffic. Moreover, particularly interesting statistics can be derived from the combined flow and packet statistics, such as the average size of flows of different kinds of traffic, and eventually how much traffic

Table 4: Top 5 Applications for All Users

Application Name	Number of Flows	% of All Flows
uTorrent	6399336	74.12
Unknown	948497	10.99
Chrome	441953	5.12
Firefox	361213	4.18
Svchost	103757	1.2

Table 5: Top 5 Applications for User 1

Application Name	Number of Flows	% of All Flows
Unknown	729868	57.56
Firefox	330498	26.07
Chrome	138105	10.89
Amule	18863	1.49
Ntpd	17929	1.41

Table 6: Top 5 Applications for User 2

Application Name	Number of Flows	% of All Flows
uTorrent	4674545	89.43
Chrome	227616	4.35
Unknown	136387	2.61
Svchost	101633	1.94
Java	38704	0.74

Table 7: Top 5 Applications for User 3

Application Name	Number of Flows	% of All Flows
uTorrent	1220728	91.36
Chrome	76062	5.69
Unknown	21764	1.63
SoftonicDownloader	15035	1.13
Java	2169	0.16

Table 8: Top 5 Applications for User 4

Application Name	Number of Flows	% of All Flows
uTorrent	504063	62.78
Moc	90358	11.25
Iexplore	64407	8.02
Unknown	60478	7.53
Firefox	26896	3.35

is created by different applications for individual users. The challenge is that it is large amounts of data, so efficient ways of handling these has to be developed.

Another important part is the recruitment of more volunteers, in order to collect larger amounts of data. Also, having appropriate background information about the users could be useful. This includes both the data about the users themselves, such as age, occupation, if the computer is shared etc., but also information about the connection, e.g. speeds and technologies.

In order to obtain more data, other researchers are invited to join the project and use it for the collection of data for scientific purposes. The code is available as open-source, and can be found together with a comprehensive documentation on our project homepage [12] located on SourceForge.

References

[1] CAIDA Internet Data – Passive Data Sources, 2012. [Online]. Available: http://www.caida.org/data/passive/.

[2] Chuck Fraleigh, Sue Moon, Bryan Lyles, Chase Cotton, Mujahid Khan, Deb Moll, Rob Rockell, Ted Seely, and S. Christophe Diot. Packet-Level Traffic Measurements from the Sprint IP Backbone. *IEEE Network*, 17(6):6–16, November 2003. DOI: 10.1109/MNET.2003.1248656.

[3] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On Dominant Characteristics of Residential Broadband Internet Traffic. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 90–102. ACM New York, Barcelona, Spain, August 2009. DOI: 10.1145/1644893.1644904.

[4] Cisco IOS NetFlow, 2012. [Online]. Available: http://www.cisco.com/go/netflow.

[5] Kartheepan Balachandran, Jacob Honoré Broberg, Kasper Revsbech, and Jens Myrup Pedersen. Volunteer-Based Distributed Traffic Data Collection System. In *Proceedings of the 12th International Conference on Advanced Communication Technology (ICACT 2010)*, volume 2, pages 1147–1152. IEEE, Phoenix Park, PyeongChang, Korea, February 2010.

[6] Kartheepan Balachandran and Jacob Honoré Broberg. Volunteer-Based Distributed Traffic Data Collection System. Master's thesis, Aalborg University, Department of Electronic Systems, Denmark, June 2010.

[7] Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for classification of traffic in computer networks. In

Proceedings of the 19th Telecommunications Forum TELFOR 2011, pages 210–213. IEEE, Belgrade, Serbia, November 2011. DOI: 10.1109/TELFOR.2011.6143528.

[8] Tomasz Bujlow, Kartheepan Balachandran, Sara Ligaard Nørgaard Hald, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for research on the Internet traffic. *TELFOR Journal*, 4(1):2–7, September 2012. Accessible: http://journal.telfor.rs/Published/Vol4No1/Vol4No1.aspx.

[9] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for Assessing Quality of Service in Broadband Networks. In *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT)*, pages 826–831. IEEE, Phoenix Park, PyeongChang, Korea, February 2012. Accessible: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6174795.

[10] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for classification of network traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of ICNC'12: 2012 International Conference on Computing, Networking and Communications (ICNC): Workshop on Computing, Networking and Communications*, pages 244–248. IEEE, Maui, Hawaii, USA, February 2012. DOI: 10.1109/ICCNC.2012.6167418.

[11] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. Classification of HTTP traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of the Fourth IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Web-based Service Architectures (PEDISWESA 2012)*, pages 882–887. IEEE, Cappadocia, Turkey, July 2012. DOI: 10.1109/ISCC.2012.6249413.

[12] Volunteer-Based System for Research on the Internet, 2012. [Online]. Available: http://vbsi.sourceforge.net/.

Paper VI

Obtaining Application-Based and Content-Based Internet Traffic Statistics

Tomasz Bujlow and Jens Myrup Pedersen

Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark
{tbu, kba, tahir, jens}@es.aau.dk

This paper is reprinted from the *Proceedings of the 6th International Conference on Signal Processing and Communication Systems (ICSPCS'12)*, pages 1–10. IEEE, Gold Coast, Queensland, Australia, December 2012.

DOI: 10.1109/ICSPCS.2012.6507984.

Abstract

Understanding Internet traffic is crucial in order to facilitate the academic research and practical network engineering, e.g. when doing traffic classification, prioritization of traffic, creating realistic scenarios and models for Internet traffic development etc. In this paper, we demonstrate how the Volunteer-Based System for Research on the Internet, developed at Aalborg University, is capable of providing detailed statistics of Internet usage. Since an increasing amount of HTTP

traffic has been observed during the last few years, the system also supports creating statistics of different kinds of HTTP traffic, like audio, video, file transfers, etc. All statistics can be obtained for individual users of the system, for groups of users, or for all users altogether. This paper presents results with real data collected from a limited number of real users over six months. We demonstrate that the system can be useful for studying the characteristics of computer network traffic in application-oriented or content-type- oriented way, and is now ready for a larger-scale implementation. The paper is concluded with a discussion about various applications of the system and the possibilities of further enhancements.

Keywords

Internet traffic, traffic classification, computer networks, per-application statistics, per-content-type statistics

1 Introduction

Monitoring traffic in computer networks and understanding the behavior of network applications is a very important challenge for both Internet Service Providers (ISPs) and scientists. ISPs focus on the business aspects of traffic monitoring, like improving the Quality of Service (QoS) in their networks. In order to setup the QoS rules in the network in a proper way, it is necessary to know what kind of traffic is flowing in the network, and how large amounts of traffic different applications account for. The knowledge of which applications are most frequently used in the network can be used by the ISPs to enhance the user experience by tuning some network parameters or setting up dedicated proxies or servers for particular applications or services. Users located in the same subnet can be compared and grouped according to their profile (like heavy user or interactive user), or distributed among the network to balance the load. Many ISPs have multiple connections to the external world, including many content deliverers. The knowledge of which connections is used most frequently can benefit in more accurate decisions from which provider the bandwidth should be bought. Finally, in many countries, the law obligates the ISPs to log all traffic, in order to be able to track down cybercrime, investigate terroristic attacks, etc. The knowledge of what the traffic is can benefit in saving storage space by logging only the important part of the traffic.

On the other hand, scientists use traffic monitoring to model traffic correctly and to create realistic scenarios of Internet usage. The models can be used for testing various options in designing networks before implementing them, examining the influence of a change in the current network design before applying it, or creating precise traffic classifiers.

There are many possibilities to obtain the relevant traffic statistics. Some data traces are available to the public (as Caida sets [1]), but most of them lack the detailed information about each packet (like payload, status of TCP flags), or about the structure of the flow (like inter-arrival times of the packets). Without the access to the real data, it would not be possible to conduct many interesting studies [2, 3]. There are, however, many possibilities to obtain the traces directly from the network by researchers. Unfortunately, this method has several drawbacks. First of all, these traces are obtained only in a few selected points, to which the researcher has access, so the traces are geographically limited. This concerns for example collecting data by Wireshark [4], or Cisco Netflow [5], which can provide some good statistics in the selected points of the network. Second, the obtained traces must be pre-classified according to the application-layer protocol, type of content carried by particular flows, etc. This task is not trivial, and it is a hard challenge to perform correct classification, especially when subject to real-time or near real-time requirements.

The simplest idea, widely used to pre-classify the traffic is using the application ports [6, 7]. Unfortunately, this fast method can be applied only to the applications or protocols, which use fixed port numbers. Nowadays, most traffic is generated by applications of Peer-to-Peer (P2P) nature, which operate on dynamic port numbers. Therefore, through port-based classification it is not possible to detect Bittorrent, or Skype [4, 8, 9].

The second commonly used solution to pre-classify data is Deep Packet Inspection (DPI). However, the name of this method can be misleading, since many DPI tools rely in fact on statistical parameters and they perform statistical classification to discover some applications. It causes some overlap and produces false positives and false negatives [10, 11]. Furthermore, DPI is quite slow and it require a lot of resources, especially processing power [4, 8]. Processing payloads of the users' data also raises privacy and confidentiality concerns [4].

To avoid the issues described above, we developed at Aalborg University a tool called Volunteer-Based System (VBS). The most articulate advantages of the system is that by monitoring at the host machines, we are able to see the traffic exactly as it is generated at the source, and we are able to see which applications are opening the sockets, enabling creating accurate mappings between applications and traffic flows. Even with a relatively low number of users, we can obtain good understanding of how different applications behave with respect to traffic, but in order to obtain the data which can be used to describe Internet usage a substantial amount of users would be needed.

This open source tool is released under *GNU General Public License v3.0* and published as a SourceForge project [12]. Both Windows and Linux versions are available. VBS is designed to collect the traffic from numerous volunteers spread around the world and, therefore, with a sufficient number of volunteers the collected data can provide us with a good statistical base. The task of the Volunteer-Based System is to collect flows

of Internet traffic data together with detailed information about each packet. The information about the application associated with each flow is taken from system sockets and appended to the flow description. Additionally, we collect the general information about the types of transferred HTTP contents, so we are able to distinguish various kinds of browser traffic. The system ideas were first described and initially implemented in [13], after which the design of our current Volunteer-Based System was described in [14]. Further improvements and refinements can be found in [15]. In parallel with our efforts [16] describes a Windows-based system which partially uses the same ideas of host based monitoring and accurate application informations. Our system was used to obtain various statistics useful for Machine Learning, Quality of Service Assessment and traffic analysis [17–19]. Our last paper [20] demonstrated how the system can be used for generating statistics at the flow level.

In this paper, we present the possibilities of the system for creating application-based or content-type-based statistics on the flow and on the packet level. It presents the results of a 6 months test study of the system, based on data from 4 users who joined at different times during this period. The main contribution is the demonstration of how the system can determine which packets are generated by which applications, and even further specify the kind of data (for example, if traffic generated by a web browser is web, audio or video traffic). The paper is organized as follows: First, in Section 2, we describe how the data is collected by the Volunteer-Based System and how the statistics are extracted. In Section 3, we present the results, and in Section 4, we conclude the paper and discuss the further work.

2 Collecting Data by Volunteer-Based System

This section presents the brief overview of VBS. We tried to highlight parts which are relevant for collecting network data and associating it with particular applications and HTTP content-types. For more details about the design and implementation of VBS, please refer to our previous paper [15].

The Volunteer-Based system is built using the client-server architecture. Clients are installed among machines belonging to volunteers, while the server is installed on the computer located at Aalborg University. Each client registers information about the data passing computer's network interfaces. Captured packets are grouped into flows. A flow is defined as a group of packets which have the same local and remote IP addresses, local and remote ports, and using the same transport layer protocol. For every flow the client registers: anonymized identifier of the client, start timestamp of the flow, anonymized local and remote IP addresses, local and remote ports, transport protocol, anonymized global IP address of the client, and name of the application associated with that flow. The name of the application is taken from the system sockets. For every packet, the client additionally registers: direction, size, state of all TCP flags (for TCP connections only), time in microseconds elapsed from the previous packet in the flow,

and type of transmitted HTTP content. We do not inspect the payload - the type of the HTTP content is obtained from the HTTP header, which is present in the first packet carrying this specific content. One HTTP flow (for example a connection to a web server) can carry multiple files: HTML documents, JPEG images, CSS stylesheets, etc. Thanks to that ability implemented in our VBS, we are able to split the flow and separate particular HTTP contents. The data collected by VBS are stored in a local file and periodically sent to the server. The task of the server is to receive the data from clients and to store them into the MySQL database.

The purpose of this study was to demonstrate the usage, so we focused on obtaining and presenting the data from a limited number of users. In future, we plan to make more wide-scale experiments. The statistics used in this paper were obtained from 4 users during the period from January to May 2012. However, the clients join VBS at different time points. The four users can be described as follows:

- User 1 - Private user in Denmark, joined the system on December 28, 2011
- User 2 - Private user in Poland, joined the system on December 28, 2011
- User 3 - Private user in Poland, joined the system on December 31, 2011
- User 4 - Private user in Denmark, joined the system on April 24, 2012

Our system was designed not only to store the complete knowledge of users' traffic in the Aalborg University database, but also to provide numerous useful statistics. These statistics can be calculated altogether, or grouped on a per-user basis, per-application basis, per-content-type basis, or on a mix of these. Most of them can be also calculated on a per-flow basis, and, therefore, they can be a direct input to various classification and clustering Machine Learning Algorithms. The calculated statistics include (but they are not limited to):

- Number of flows
- Percent of all number of flows
- Average flow duration (in seconds)
- Average number of packets in flow
- Percent of inbound packets in flow
- Average inbound, outbound, and total packet size
- Minimum inbound, outbound, and total packet size
- Maximum inbound, outbound, and total packet size

- Median of inbound, outbound, and total packet size
- First quartile of inbound, outbound, and total packet size
- Third quartile of inbound, outbound, and total packet size
- Standard deviation of inbound, outbound, and total packet size
- Percent of inbound, outbound, and total packets which carry data
- Percent of data packets which are inbound
- Percent of inbound, outbound, and total data packets which are small (below 70 B)
- Percent of small data packets which are inbound
- Percent of inbound, outbound, and total data packets which are big (above 1320 B)
- Percent of big data packets which are inbound
- Percent of inbound, outbound, and total packets which have ACK flag
- Percent of packets with ACK flag which are inbound
- Percent of inbound, outbound, and total packets which have PSH flag
- Percent of packets with PSH flag which are inbound
- Amount of traffic (in Megabytes)
- Percent of traffic which is inbound
- Percent of traffic from all flows
- Number of TCP, UDP, and HTTP flows
- Amount of traffic (in Megabytes) carried by TCP, UDP, and HTTP flows

Due to limited length of this paper, we are not able to present and describe all the generated statistics. Instead, we decided to focus on a few per-application and per-content-type measurements, which we obtained for all users separately and altogether.

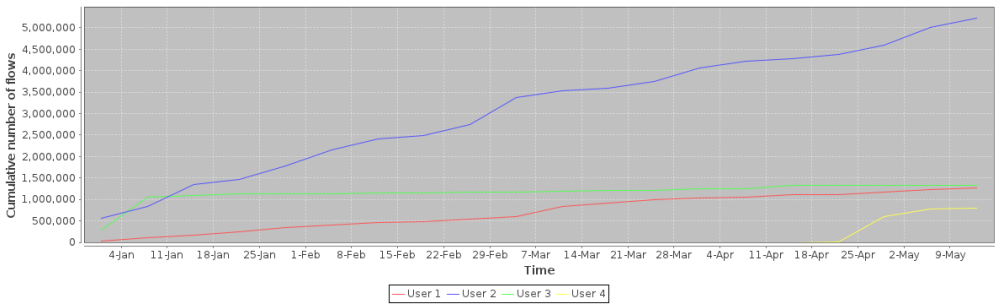

Figure 1: Cumulative Number of Flows Belonging to Different Users Over Time

3 Results

3.1 Number of Flows vs Number of Bytes

The amount of traffic passing a network connection can be characterized using various metrics. The most common used are number of bytes and number of flows. They are dependent on each other, because the increasing number of network flows always increase the number of transferred bytes. However, flows can be short or long, and packets belonging to that flows can have various lengths. Therefore, on different machines, the increase of number of flows have different impact on the increase of number of Bytes. The cumulative numbers of flows collected over the time from different machines are shown in Figure 1. Similarly, the cumulative numbers of bytes collected from the same clients over the same period of time are shown in Figure 2. The characteristics are quite similar - the difference concerns the client number 3. This user produces higher number of flows than users 1 and 4, but it generates the lowest traffic among all the users. It means that the user number 3 must use more interactive applications (producing smaller packets) than the other users, or use applications producing shorter flows. Based on that we can assume that this user is not a heavy downloader – file downloads usually use only a few flows, but each of them carries large amounts of data. Our suspicions will be proved in the next points, when we show the distribution of different applications among all the observed users.

3.2 Top 10 Applications

Analyzing the network traffic in the application-wise way is a very challenging task. Our Volunteer-Based System (VBS) is able to associate each flow with the application name, which is taken from the system sockets. This approach is quite straightforward, but unfortunately it also has one big drawback - the socket must be open for a sufficiently long time to allow VBS to notice it and to grab the application name. Consequently, a

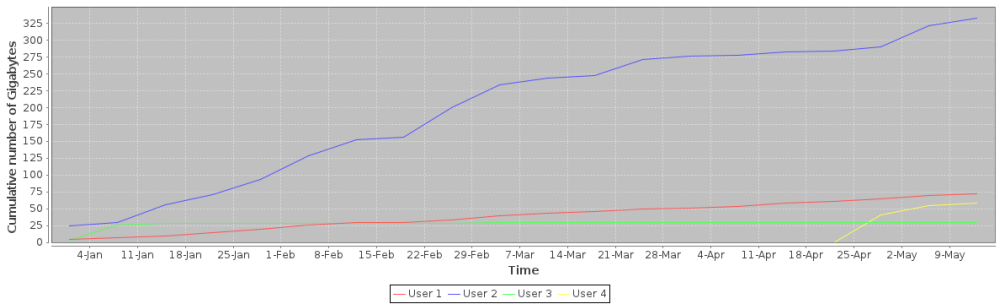

Figure 2: Cumulative Amount of Traffic Belonging to Different Users Over Time

Table 1: Top 10 Applications for All Users

Order Number	Application	Amount [MB]	% of All Traffic	Number of Flows	% of all Flows	Packets in a Flow (Average)
1	uTorrent	348694	61	7151020	72	63
2	chrome	55675	9	599228	6	115
3	firefox	33657	5	381805	3	109
4	svchost	27994	4	118059	1	405
5	moc	20943	3	141557	1	179
6	java	18767	3	81379	0	280
7	libgcflashplay	12028	2	88	0	139567
8	libgcflashpla	8312	1	59	0	135731
9	Unknown	7395	1	1135040	11	11
10	SoftonicDownloader	7105	1	15035	0	509

substantial number of short flows lack the associated application. During our research, 260 different process names accounting for 556.7 GB of data were identified, and for this study we just list the top 10 of them. To emphasize the influence of the flow length on the ability to obtain the application name, the average number of packets in flow is also included in these statistics.

The top application names for all users altogether are shown in Table 1. We also include the information about the number of flows belonging to each application. The applications are ordered according to the amounts of transmitted data.

The obtained results show that:

- The average number of packets in flows without assigned application name is 11, comparing to 63–139567 in flows with the application name assigned. This confirms that our VBS is not good in providing application names for short flows. However, it is worth noticing that flows without assigned application name account only for 1 % of the whole traffic volume.

- A big number of flows does not mean big amount of data. Flows without assigned application name account for 11 % of of total number of flows (second position), but only for 1 % of the whole traffic volume (9. position).
- Applications having large number of packets in a flow (like *libgcflashplay* and *libgcflashplaya*, responsible for streaming video through web browser) can account for more traffic than applications having small number of flows (like all applications belonging to the *Unknown* group together). Normally, it would not be surprising, but in this case the proportion of the number of flows belonging to *libgcflashplay* to the *Unknown* group is 1:12898.

The same statistics for individual users are shown in Tables 2–5. The results also depict the reason for the phenomena described earlier in this paper. Most flows (91 %) belonging to user number 3 consist of 23 packets in average, comparing to 70–112 packets for other users. That is why we encounter more flows from user number 3 than from users 1 and 4, but we see lower amount of traffic.

The cumulative amount of traffic generated by top 5 applications for all users altogether are shown in Figure 3. It is clearly depicted that the amount of traffic generated by *uTorrent* is over 6 times bigger than the amount of traffic generated by the second biggest traffic provider in our chart (*chrome*), and 3 times bigger than the amount of traffic generated by all applications besides the top 5. Large amount of *uTorrent* traffic led us to study its behavior more carefully. The cumulative amounts of traffic generated by downloading and uploading files by *uTorrent* are shown in Figure 4 and Figure 5, respectively. The charts depict the very interesting characteristics of bittorrent traffic - downloading and uploading is realized simultaneously, so the download and upload curves have the same shapes. It is, however, worth noticing that the amount of downloaded traffic is around 7 times bigger than the amount of traffic uploaded by the clients. The next interesting observation is that user number 4 uploads almost the same amount

Table 2: Top 10 Applications for User 1

Order Number	Application	Amount [MB]	% of All Traffic	Number of Flows	% of all Flows	Packets in a Flow (Average)
1	firefox	29921	28	330734	21	112
2	chrome	27143	26	242937	15	142
3	libgcflashplay	12028	11	88	0	139567
4	libgcflashplaya	8312	8	59	0	135731
5	Unknown	5449	5	869134	56	9
6	http	4718	4	3104	0	1520
7	plugin-contain	2632	2	413	0	8058
8	iplalite	2618	2	473	0	5661
9	clwb3	2525	2	266	0	11300
10	filezilla	2249	2	62	0	38528

Table 3: Top 10 Applications for User 2

Order Number	Application	Amount [MB]	% of All Traffic	Number of Flows	% of all Flows	Packets in a Flow (Average)
1	uTorrent	295981	80	5379088	89	70
2	svchost	27757	7	115166	1	413
3	chrome	25703	6	272772	4	112
4	java	14746	3	42703	0	417
5	firefox	1660	0	5144	0	354
6	Unknown	1154	0	152489	2	17
7	skype	842	0	18452	0	307
8	thebat	339	0	1908	0	238
9	SoftwareUpdate	223	0	32	0	7327
10	dropbox	145	0	7106	0	74

Table 4: Top 10 Applications for User 3

Order Number	Application	Amount [MB]	% of All Traffic	Number of Flows	% of all Flows	Packets in a Flow (Average)
1	uTorrent	19315	62	1267869	91	23
2	SoftonicDownloader	7105	22	15035	1	509
3	chrome	2819	9	83349	5	46
4	java	1524	4	2214	0	849
5	svchost	121	0	237	0	550
6	Unknown	66	0	24197	1	10
7	Pity	28	0	33	0	914
8	e-pity2011	20	0	12	0	1800
9	AcroRd32	1	0	15	0	94
10	AdobeARM	0	0	11	0	31

Table 5: Top 10 Applications for User 4

Order Number	Application	Amount [MB]	% of All Traffic	Number of Flows	% of all Flows	Packets in a Flow (Average)
1	uTorrent	33395	50	504063	52	78
2	moc	20943	31	141557	14	179
3	iexplore	3760	5	81306	8	60
4	java	2494	3	36459	3	86
5	firefox	2074	3	45927	4	59
6	vmnat	2015	3	2983	0	709
7	Unknown	722	1	89220	9	15
8	mantra	259	0	7746	0	49
9	javaw	160	0	158	0	1420
10	svchost	113	0	2656	0	51

Table 6: Top 10 HTTP Content-Types for All Users

Order Number	Content-Type	Amount [MB]	% of All HTTP Traffic	Number of Contents	% of All	Average Number of Packets
1	video/x-flv	35828	34	16238	0	1543
2	audio/mpeg	11884	11	1945	0	4355
3	application/octet-stream	8832	8	17688	0	351
4	application/x-msdos-program	7095	6	1673	0	2963
5	video/mp4	5987	5	5983	0	696
6	image/jpeg	5888	5	516090	18	9
7	application/x-debian-package	5119	4	2444	0	1447
8	application/zip	3278	3	309	0	7426
9	text/html	2398	2	618695	22	4
10	text/plain	2013	2	348826	12	6

of data as it downloads, so it is possible that he has a symmetric Internet connection.

3.3 Top 10 HTTP Content-Types

The previous subsection showed that besides the bittorrent traffic, web browsers account for the most of traffic transmitted in computer networks. This fact is not surprising since more and more services are becoming web-based, including web radio, web television, web applications, etc. Therefore, the knowledge of which application generated the traffic is not sufficient and we needed to perform the examination what the browser traffic is. Our Volunteer-Based System is able to provide us information about the *Content-Type* headers transmitted by the web server to the browser for each part of information received by the client. During our research, 191 different HTTP content-types accounting for 98.5 GB of data were identified, and for this study we just list the top 10 of them. Grouping such content-types into particular categories (like audio, video, binary data, etc) is outside the scope of this paper and it is a subject to further examinations.

The top HTTP content-types for all users altogether are shown in Table 6. The content types are ordered according to the amounts of transmitted data. Unlikely than when describing traffic generated by various applications (we were taking into account inbound as well as outbound traffic), we consider in this point only the inbound traffic. The reason is that only the inbound traffic is responsible for delivering the content to the clients. The outbound traffic while transmitting HTTP contents is very low and it consists of small packets containing acknowledgments and new parts requests. Table 7 contains the comparison of the inbound and outbound characteristics of the traffic while downloading particular contents via HTTP.

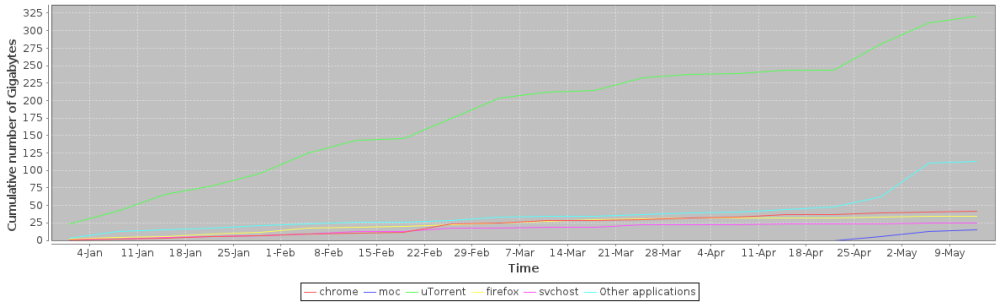

Figure 3: Cumulative Amount of Traffic Generated by Top 5 Applications Over Time

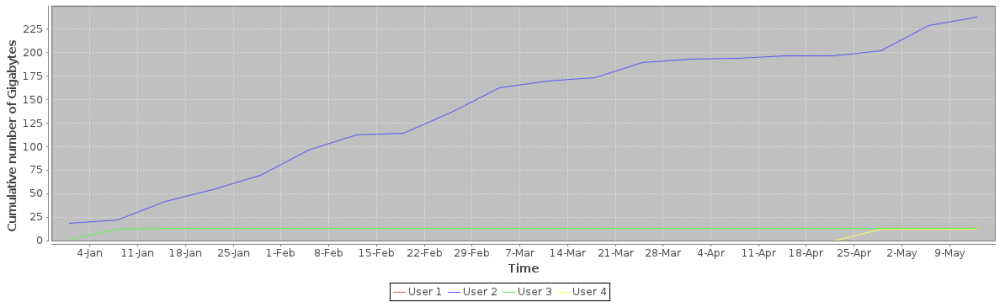

Figure 4: Cumulative Amount of Traffic Downloaded by *uTorrent* Over Time

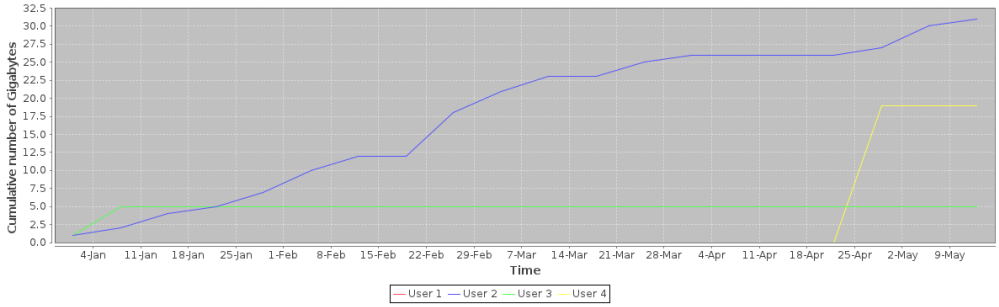

Figure 5: Cumulative Amount of Traffic Uploaded by *uTorrent* Over Time

Table 7: Characteristics of Inbound and Outbound Traffic for Top 10 HTTP Content-Types for All Users

Order Number	Content-Type	Average Inbound Packet Size [B]	Average Outbound Packet Size [B]	% of Inbound Packets	% of Inbound Bytes
1	video/x-flv	1499	51	64	98
2	audio/mpeg	1470	48	57	97
3	application/octet-stream	1474	46	64	98
4	application/x-msdos-program	1498	41	65	98
5	video/mp4	1244	104	58	94
6	image/jpeg	1496	47	66	98
7	application/x-debian-package	1516	48	67	98
8	application/zip	1496	41	65	98
9	text/html	880	226	53	81
10	text/plain	1032	149	63	92

The results shows that the majority of HTTP traffic is generated by video and binary files downloaded by users. The web traffic, however, also occupies three places (*image/jpeg*, *text/html*, and *text/plain*) in the list of top 10 HTTP content-types. It is worth mentioning that these three content-types account for 52 % of the total number of transferred HTTP contents, but only for 9 % of the total number of transferred HTTP traffic, due to a low number of packets from which these contents consist of (4–9 in average).

The same statistics for individual users are shown in Tables 8–11. For each user, in the top 10 content-types we can find the ones characteristic for web browsing activities (*image/jpeg*, *text/html*, and *text/plain*) and the ones characteristic for video services, as *YouTube* (*video/x-flv*). The latter content-type is also commonly used by Video on Demand (VoD) applications, as *Ipla*, which are not web browsers, but they use HTTP to download video data to the user's computer. The other interesting dependency, which can be noticed based on these four tables is the inverse proportionality between the number of observed occurrences of the particular content-type and the average number of packets contained by the content. It means that most often we observe relatively short contents (as HTML files or web images), and larger ones are more rare (as movies).

Table 8: Top 10 HTTP Content-Types for User 1

Order Number	Content-Type	Amount [MB]	% of All HTTP Traffic	Number of Contents	% of All	Average Number of Packets
1	video/x-flv	23757	45	6490	0	2554
2	audio/mpeg	6693	12	198	0	24380
3	video/mp4	3428	6	550	0	4316
4	application/x-debian-package	3319	6	202	0	11190
5	image/jpeg	3019	6	271485	21	9
6	application/octet-stream	2401	4	6636	0	261
7	text/html	1184	2	232431	18	5
8	text/plain	1156	2	168202	13	6
9	video/webm	807	1	30	0	18612
10	image/png	781	1	86470	6	8

Table 9: Top 10 HTTP Content-Types for User 2

Order Number	Content-Type	Amount [MB]	% of All HTTP Traffic	Number of Contents	% of All	Average Number of Packets
1	video/x-flv	5553	35	6210	0	632
2	video/mp4	2083	13	5311	0	276
3	application/octet-stream	1840	11	6777	0	194
4	image/jpeg	1473	9	111872	14	11
5	text/html	492	3	133701	17	5
6	text/plain	442	3	139826	18	4
7	application/rar	431	2	2	0	152136
8	image/png	429	2	43620	5	9
9	application/x-shockwave-flash	404	2	11304	1	27
10	application/x-javascript	298	2	39194	5	7

Table 10: Top 10 HTTP Content-Types for User 3

Order Number	Content-Type	Amount [MB]	% of All HTTP Traffic	Number of Contents	% of All	Average Number of Packets
1	application/x-msdos-program	6913	71	1440	1	3366
2	video/x-flv	1264	12	560	0	1604
3	image/jpeg	195	2	21811	15	9
4	application/x-shockwave-flash	173	1	3583	2	36
5	video/mp4	157	1	33	0	3373
6	image/png	148	1	21156	14	7
7	application/x-javascript	141	1	16095	11	8
8	application/x-compress	141	1	1	0	99186
9	application/octet-stream	101	1	313	0	229
10	text/html	87	0	26875	18	4

Table 11: Top 10 HTTP Content-Types for User 4

Order Number	Content-Type	Amount [MB]	% of All HTTP Traffic	Number of Contents	% of All	Average Number of Packets
1	audio/mpeg	5027	20	925	0	3808
2	video/x-flv	5005	20	2978	0	1185
3	application/octet-stream	4463	17	3962	0	788
4	application/zip	3062	12	123	0	17419
5	application/x-debian-package	1797	7	2242	0	560
6	image/jpeg	1169	4	110922	18	9
7	application/x-gzip	646	3	37	0	12700
8	text/html	642	3	225688	37	3
9	text/plain	381	1	37874	6	9
10	video/mp4	316	1	89	0	2495

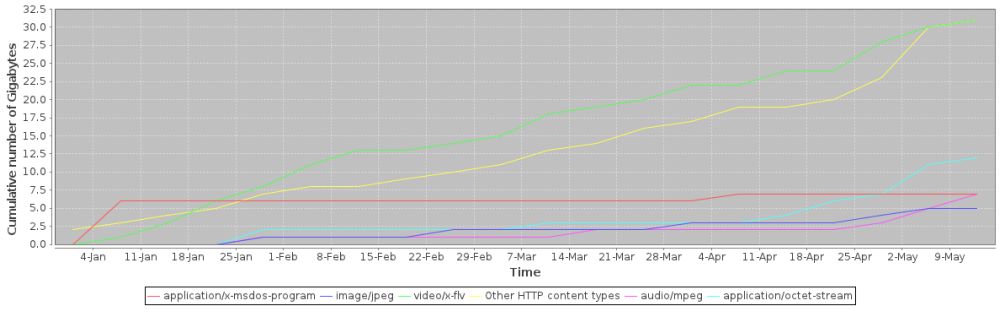

Figure 6: Cumulative Amount of Traffic Generated by Top 5 HTTP Content-Types

The cumulative amount of traffic generated by the top 5 HTTP content-types for all users altogether are shown in Figure 6. It is clearly depicted that the amount of traffic generated by *video/x-flv* is around 2.5 times bigger than the amount of traffic generated by the second biggest traffic provider in our chart (*application/octet-stream*), and it also corresponds to the amount of traffic generated by all HTTP content-types besides the top 5.

3.4 Characterizing Application Traffic

With the data collected it is possible to characterize traffic from different applications by a large number of metrics. In this section, we will shortly demonstrate some of the interesting metrics, which can be used to characterize traffic based on the data collected throughout the study:

- Average packet sizes: inbound, outbound, and total
- Distribution of inbound and outbound packets
- Distribution of inbound and outbound packets carrying data

The results are presented in Table 12. Quite a few interesting observations can be made. While not surprising, it is interesting to observe that for *chrome* 60 % of the packets are inbound. If only packets carrying data are taken into account, this number increases to 71%. For *dropbox*, which was extensively analyzed on the flow and volume level in [21], it is interesting to note that while the number of inbound and outbound packets are approximately the same, there is actually quite a large difference in the size of inbound and outbound packets. Looking into the more detailed figures for *dropbox*, it can actually be seen that 49 % of the outbound data packets are big (above 1320 B), while this is only so for 12 % of the inbound data packets.

Table 12: Characterizing Traffic Generated by Various 5 Applications for All Users

Application Name	Average Packet Size In [B]	Average Packet Size Out [B]	Average Packet Size [B]	% of Packets In	% of Packets Out	% of Data Packets In	% of Data Packets Out
chrome	1314	130	842	60	40	71	29
dropbox	272	832	562	48	52	43	57
skype	207	178	193	51	49	51	49
uTorrent	1133	351	810	58	42	61	39
wget	1501	54	864	55	45	55	45

4 Conclusion and Discussion

In this paper, we have demonstrated how the Volunteer-Based System developed at Aalborg University can be used for generating useful statistics of Internet traffic usage – statistics which are useful academic as well as practical network engineering purposes. The system is based on monitoring traffic on the host, which has several advantages over traditional approaches for traffic monitoring - in particular, it is possible to obtain precise mappings between the applications and the traffic generated, which is a big help when training statistical classifiers. The paper demonstrated some of the statistics that can be obtained using the system, and examples of how they can be further processed to useful statistical information.

We focused in our studies on statistics calculated for various network applications, and presented both the overall statistics and statistics that characterizes specific applications. Web browsers can carry today many different kinds of traffic, including interactive voice and video. We have demonstrated how we can use VBS to separate various types of HTTP traffic. The information gathered by the system can be used in many different ways: to create realistic models of computer networks, to provide accurate training data to Machine Learning Algorithms, to develop new and enhance existing networks. The current study has involved only a low number of volunteers in order to test the system prior to a large-scale implementation, and with the satisfactory results we are now ready to move on.

This paper is mainly intended as a demonstration of the system, and with the limited number of users the results do not represent the truth of neither application behavior nor distribution between applications. For the latter, it would be necessary to recruit not only a large number of volunteers, but also a set of volunteers representing the group that should be studied. For the former, a smaller number of users would be sufficient - as long as the group is large enough to ensure that different usages of the different applications are covered. That being said, we still believe that the results provide interesting indications of application behaviors for the most common applications such as Bittorrent and web traffic. It is important to keep in mind, though, that different user groups would still have different behaviors - for example, the use of e.g. web radios

or web browsers could be different in different countries/regions depending on cultures, as well as between different user segments. If the system is to be used for training statistical classifiers, we would recommend that the data are collected from the same network as the classifier is later to be used in, in order to cope with these challenges.

Being aware that the amount of data is crucial, we highly encourage user groups and researchers to use the proposed system for collecting data and if possible sharing the anonymized traces with other researchers. Therefore, the system is based on open source code available from [12]. Other contributors would be welcome to set up their own servers for data collection, or to collaborate with the authors on the data collection.

Future research will focus on grouping the applications and the HTTP content-types into several sets, like voice, video, file transfer, interactive browsing, etc. This is not a trivial task, since grouping manually such large number of applications and content-types is not doable. Furthermore, the interactive connections (like interactive web browsing) should treat all the files in that connections as a whole, without splitting that into particular HTML documents, web images, stylesheets, etc. A kind of clustering algorithm can be used to partially automatize that process.

References

[1] CAIDA Internet Data – Passive Data Sources, 2012. [Online]. Available: http://www.caida.org/data/passive/.

[2] Chuck Fraleigh, Sue Moon, Bryan Lyles, Chase Cotton, Mujahid Khan, Deb Moll, Rob Rockell, Ted Seely, and S. Christophe Diot. Packet-Level Traffic Measurements from the Sprint IP Backbone. *IEEE Network*, 17(6):6–16, November 2003. DOI: 10.1109/MNET.2003.1248656.

[3] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On Dominant Characteristics of Residential Broadband Internet Traffic. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 90–102. ACM New York, Barcelona, Spain, August 2009. DOI: 10.1145/1644893.1644904.

[4] Jun Li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. Real-time P2P traffic identification. In *Proceedings of the IEEE Global Telecommunications Conference (IEEE GLOBECOM 2008)*, pages 1–5. IEEE, New Orleans, Louisiana, USA, December 2008. DOI: 10.1109/GLOCOM.2008.ECP.475.

[5] Cisco IOS NetFlow, 2012. [Online]. Available: http://www.cisco.com/go/netflow.

[6] Riyad Alshammari and A. Nur Zincir-Heywood. Machine Learning based encrypted traffic classification: identifying SSH and Skype. In *Proceedings of the*

IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA 2009), pages 1–8. IEEE, Ottawa, Ontario, Canada, July 2009. DOI: 10.1109/CISDA.2009.5356534.

[7] Sven Ubik and Petr Žejdl. Evaluating application-layer classification using a Machine Learning technique over different high speed networks. In *Proceedings of the Fifth International Conference on Systems and Networks Communications (ICSNC)*, pages 387–391. IEEE, Nice, France, August 2010. DOI: 10.1109/ICSNC.2010.66.

[8] Ying Zhang, Hongbo Wang, and Shiduan Cheng. A Method for Real-Time Peer-to-Peer Traffic Classification Based on C4.5. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 1192–1195. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689126.

[9] Jing Cai, Zhibin Zhang, and Xinbo Song. An analysis of UDP traffic classification. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 116–119. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689203.

[10] Riyad Alshammari and A. Nur Zincir-Heywood. Unveiling Skype encrypted tunnels using GP. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, Barcelona, Spain, July 2010. DOI: 10.1109/CEC.2010.5586288.

[11] L7-filter Supported Protocols, 2012. [Online]. Available: `http://l7-filter.sourceforge.net/protocols`.

[12] Volunteer-Based System for Research on the Internet, 2012. [Online]. Available: `http://vbsi.sourceforge.net/`.

[13] Kartheepan Balachandran, Jacob Honoré Broberg, Kasper Revsbech, and Jens Myrup Pedersen. Volunteer-Based Distributed Traffic Data Collection System. In *Proceedings of the 12th International Conference on Advanced Communication Technology (ICACT 2010)*, volume 2, pages 1147–1152. IEEE, Phoenix Park, PyeongChang, Korea, February 2010.

[14] Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for classification of traffic in computer networks. In *Proceedings of the 19th Telecommunications Forum TELFOR 2011*, pages 210–213. IEEE, Belgrade, Serbia, November 2011. DOI: 10.1109/TELFOR.2011.6143528.

[15] Tomasz Bujlow, Kartheepan Balachandran, Sara Ligaard Nørgaard Hald, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for research on the Internet traffic. *TELFOR Journal*, 4(1):2–7, September 2012. Accessible: `http://journal.telfor.rs/Published/Vol4No1/Vol4No1.aspx`.

[16] Runyuan Sun, Bo Yang, Lizhi Peng, Zhenxiang Chen, Lei Zhang, and Shan Jing. Traffic Classification Using Probabilistic Neural Networks. In *Proceedings of the Sixth International Conference on Natural Computation (ICNC 2010)*, volume 4, pages 1914–1919. IEEE, Yantai, Shandong, China, August 2010. DOI: 10.1109/ICNC.2010.5584648.

[17] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for classification of network traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of ICNC'12: 2012 International Conference on Computing, Networking and Communications (ICNC): Workshop on Computing, Networking and Communications*, pages 244–248. IEEE, Maui, Hawaii, USA, February 2012. DOI: 10.1109/ICCNC.2012.6167418.

[18] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for Assessing Quality of Service in Broadband Networks. In *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT)*, pages 826–831. IEEE, Phoenix Park, PyeongChang, Korea, February 2012. Accessible: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6174795.

[19] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. Classification of HTTP traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of the Fourth IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Web-based Service Architectures (PEDISWESA 2012)*, pages 882–887. IEEE, Cappadocia, Turkey, July 2012. DOI: 10.1109/ISCC.2012.6249413.

[20] Jens Myrup Pedersen and Tomasz Bujlow. Obtaining Internet Flow Statistics by Volunteer-Based System. In *Fourth International Conference on Image Processing & Communications (IP&C 2012), Image Processing & Communications Challenges 4, AISC 184*, pages 261–268. Springer Berlin Heidelberg, Bydgoszcz, Poland, September 2012. DOI: 10.1007/978-3-642-32384-3_32.

[21] Idilio Drago, Marco Mellia, Maurizio M Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside Dropbox: Understanding Personal Cloud Storage Services. In *Proceedings of the 2012 ACM Internet Measurement Conference (IMC '12)*, pages 481–494. ACM, Boston, Massachusetts, USA, November 2012. DOI: 10.1145/2398776.2398827.

Paper VII

Is our Ground-Truth for Traffic Classification Reliable?

Valentín Carela-Español[a], Tomasz Bujlow[b], and Pere Barlet-Ros[a]

[a] *Broadband Communications Research Group, Department of Computer Architecture, Universitat Politècnica de Catalunya, ES-08034, Barcelona, Spain*
{vcarela, pbarlet}@ac.upc.edu

[b] *Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark*
tbu@es.aau.dk

This paper[1] is reprinted from the *Proceedings of the 15th Passive and Active Measurement Conference (PAM 2014), Proceedings Series: Lecture Notes in Computer Science 8362*, pages 98–108. Springer International Publishing Switzerland, Los Angeles, USA, March 2014.

DOI: 10.1007/978-3-319-04918-2_10.

[1] This research was funded by the Spanish Ministry of Economy and Competitiveness under contract TEC2011-27474 (NOMADS project), the Comissionat per a Universitats i Recerca del DIUE de la Generalitat de Catalunya (ref. 2009SGR-1140), and by the European Regional Development Fund (ERDF).

Abstract

The validation of the different proposals in the traffic classification literature is a controversial issue. Usually, these works base their results on a ground-truth built from private datasets and labeled by techniques of unknown reliability. This makes the validation and comparison with other solutions an extremely difficult task. This paper aims to be the first step towards addressing the validation and trustworthiness problem of network traffic classifiers. We perform a comparison between 6 well-known DPI-based techniques, which are frequently used in the literature for ground-truth generation. In order to evaluate these tools, we have carefully built a labeled dataset of more than 500 000 flows, which contains traffic from popular applications. Our results present *PACE*, a commercial tool, as the most reliable solution for ground-truth generation. However, among the open-source tools available, *nDPI* and especially *Libprotoident*, also achieve very high precision, while other, more frequently used tools (e.g., *L7-filter*) are not reliable enough and should not be used for ground-truth generation in their current form.

Keywords

accuracy, PACE, OpenDPI, nDPI, Libprotoident, NBAR, L7-filter

1 Introduction and Related Work

During the last decade, traffic classification has considerably increased its relevance, becoming a key aspect for many network related tasks. The explosion of new applications and techniques to avoid detection (e.g., encryption, protocol obfuscation) have substantially increased the difficulty of traffic classification. The research community have thrown itself into this problem by proposing many different solutions. However, this problem is still far from being solved [1].

Most traffic classification solutions proposed in the literature report very high accuracy. However, these solutions mostly base their results on a private ground-truth (i.e., dataset), usually labeled by techniques of unknown reliability (e.g., ports-based or DPI-based techniques [2–5]). That makes it very difficult to compare and validate the different proposals.

The use of private datasets is derived from the lack of publicly available datasets with payload. Mainly because of privacy issues, researchers and practitioners are not allowed to share their datasets with the research community. To the best of our knowledge, just one work has tackled this problem. Gringoli et al. in [6] published anonymized traces without payload, but accurately labeled using GT. This dataset is very interesting to evaluate Machine Learning-based classifiers, but the lack of payload makes it unsuitable for DPI-based evaluation.

Table 1: DPI-Based Techniques Evaluated

Name	Version	Applications
PACE	1.41 (June 2012)	1000
OpenDPI	1.3.0 (June 2011)	100
nDPI	rev. 6391 (March 2013)	170
L7-filter	2009.05.28 (May 2009)	110
Libprotoident	2.0.6 (Nov 2012)	250
NBAR	15.2(4)M2 (Nov 2012)	85

Another crucial problem is the reliability of the techniques used to set the ground-truth. Most papers show that researchers usually obtain their ground-truth through port-based or DPI-based techniques [2–5]. The poor reliability of port-based techniques is already well known, given the use of dynamic ports or well-known ports of other applications [7, 8]. Although the reliability of DPI-based techniques is still unknown, according to conventional wisdom they are, in principle, one of the most accurate techniques.

Some previous works evaluated the accuracy of DPI-based techniques [3, 5, 9, 10]. These studies rely on a ground-truth generated by another DPI-based tool [5], port-based technique [3] or a methodology of unknown reliability [9, 10], making their comparison and validation very difficult. Recently, a concomitant study to ours [10] compared the performance of four DPI-based techniques (i.e., L7-filter, Tstat, nDPI, and Libprotoident). This parallel study confirms some of the findings of our work presenting nDPI and Libprotoident as the most accurate open-source DPI-based techniques. In [11] the reliability of L7-filter and a port-based technique was compared using a dataset obtained by GT [6] showing that both techniques present severe problems to accurately classify the traffic.

This paper presents two main contributions. First, we publish a reliable labeled dataset with full packet payloads [12]. The dataset has been artificially built in order to allow us its publication. However, we have manually simulated different behaviors to make it as representative as possible. We used VBS [13] to guarantee the reliability of the labeling process. This tool can label the flows with the name of the process that created them. This allowed us to carefully create a reliable ground-truth that can be used as a reference benchmark for the research community. Second, using this dataset, we evaluated the performance and compared the results of 6 well-known DPI-based techniques, presented in Table 1, which are widely used for the ground-truth generation in the traffic classification literature.

These contributions pretend to be the first step towards the impartial validation of network traffic classifiers. They also provide to the research community some insights about the reliability of different DPI-based techniques commonly used in the literature for ground-truth generation.

2 Methodology

2.1 Testbed

Our testbed is based on VMWare virtual machines (VMs). We installed three VMs for our data generating stations and we equipped them with Windows 7 (W7), Windows XP (XP), and Ubuntu 12.04 (LX). Additionally, we installed a server VM for data storage. To collect and accurately label the flows, we adapted Volunteer-Based System (VBS) developed at Aalborg University [13]. The task of VBS is to collect information about Internet traffic flows (i.e., start time of the flow, number of packets contained by the flow, local and remote IP addresses, local and remote ports, transport layer protocol) together with detailed information about each packet (i.e., direction, size, TCP flags, and relative timestamp to the previous packet in the flow). For each flow, the system also collects the process name associated with that flow. The process name is obtained from the system sockets. This way, we can ensure the application associated to a particular traffic. Additionally, the system collects some information about the HTTP content type (e.g., *text/html*, *video/x-flv*). The captured information is transmitted to the VBS server, which stores the data in a MySQL database. The design of VBS was initially described in [13]. On every data generating VM, we installed a modified version of VBS. The source code of the modified version was published in [14] under a *GPL license*. The modified version of the VBS client captures full Ethernet frames for each packet, extracts HTTP *URL* and *Referer* fields. We added a module called *pcapBuilder*, which is responsible for dumping the packets from the database to PCAP files. At the same time, INFO files are generated to provide detailed information about each flow, which allows us to assign each packet from the PCAP file to an individual flow. We also added a module called *logAnalyzer*, which is responsible for analyzing the logs generated by the different DPI tools, and assigning the results of the classification to the flows stored in the database.

2.2 Selection of the Data

The process of building a representative dataset, which characterizes a typical user behavior, is a challenging task, crucial on testing and comparing different traffic classifiers. Therefore, to ensure the proper diversity and amount of the included data, we decided to combine the data on a multidimensional level. Based on w3schools statistics, we selected Windows 7 (55.3 % of all users), Windows XP (19.9 %), and Linux (4.8 %) – state for January 2013. Apple computers (9.3 % of overall traffic) and mobile devices (2.2 %) were left as future work. The selected applications are shown below.

- Web browsers: based on w3schools statistics: Chrome and Firefox (W7, XP, LX), Internet Explorer (W7, XP).

- BitTorrent clients: based on CNET ranking: uTorrent and Bittorrent (W7, XP), Frostwire and Vuze (W7, XP, LX)
- eDonkey clients: based on CNET ranking: eMule (W7, XP), aMule (LX)
- FTP clients: based on CNET ranking: FileZilla (W7, XP, LX), SmartFTP Client (W7, XP), CuteFTP (W7, XP), WinSCP (W7, XP)
- Remote Desktop servers: built-in (W7, XP), xrdp (LX)
- SSH servers: sshd (LX)
- Background traffic: DNS and NTP (W7, XP, LX), NETBIOS (W7, XP)

The list of visited websites was based on the top 500 websites according to Alexa statistics. We chose several of them taking into account their rank and the nature of the website (e.g., search engines, social medias, national portals, video websites) to assure the variety of produced traffic. These websites include: Google, Facebook, YouTube, Yahoo!, Wikipedia, Java, and Justin.tv. For most websites, we performed several random clicks to linked external websites, which should better characterize the real behavior of the real users and include also other websites not included in the top 500 ranking. This also concerns search engines, from which we manually generated random clicks to the destination web sites. Each of the chosen websites was processed by each browser. In case it was required to log into the website, we created fake accounts. In order to make the dataset as representative as possible we have simulated different human behaviors when using these websites. For instance, on Facebook, we log in, interact with friends (e.g., chat, send messages, write in their walls), upload pictures, create events or play games. On YouTube, we watched the 10 most popular videos, which we randomly paused, resumed, and rewound backward and forward. Also, we randomly made some comments and clicked *Like* or *Not like* buttons. The detailed description of actions performed with the services is listed in our technical report [15]. We tested the P2P (BitTorrent and eDonkey) clients by downloading files of different sizes and then leaving the files to be seeded for some time, in order to obtain enough of traffic in both directions. We tried to test every FTP client using both the active transfer mode (PORT) and passive transfer mode (PASV), if the client supports such a mode.

2.3 Extracting the Data for Processing

Each DPI tool can have different requirements and features, so the extracting tool must handle all these issues. The PCAP files provided to PACE, OpenDPI, L7-filter, nDPI, and Libprotoident are accompanied by INFO files, which contain the information about the start and end of each flow, together with the flow identifier. Because of that, the

software, which uses the DPI libraries, can create and terminate the flows appropriately, as well as provide the classification results together with the flow identifier. Preparing the data for NBAR classification is more complicated. There are no separate INFO files describing the flows, since the classification is made directly on the router. We needed to extract the packets in a way that allows the router to process and correctly group them into flows. We achieved that by changing both the source and destination MAC addresses during the extraction process. The destination MAC address of every packet must match up with the MAC address of the interface of the router, because the router cannot process any packet which is not directed to its interface on the MAC layer. The source MAC address was set up to contain the identifier of the flow to which it belongs, so the flows were recognized by the router according to our demands. To the best of our knowledge, this is the first work to present a scientific performance evaluation of NBAR.

2.4 Classification Process

We designed a tool, called *dpi_benchmark*, which can read the PCAP files and provide the packets one-by-one to PACE, OpenDPI, L7-filter, nDPI, and Libprotoident. All the flows are started and terminated based on the information from the INFO files. After the last packet of the flow is sent to the classifier, the tool obtains the classification label associated with that flow. The labels are written to the log files together with the flow identifier, which makes us later able to relate the classification results to the original flows in the database. A brief description of the DPI-tools used in this study is presented in Table 1. Although some of the evaluated tools have multiple configuration parameters, we have used in our evaluation the default configuration for most of them. A detailed description of the evaluated DPI-tools and their configurations can be found in [15].

Classification by NBAR required us to set up a full working environment. We used GNS3 – a graphical framework, which uses Dynamips to emulate our Cisco hardware. We emulated the 7200 platform, since only for this platform supported by GNS3 was available the newest version of Cisco IOS (version 15), which contains Flexible NetFlow. The router was configured by us to use Flexible NetFlow with NBAR on the created interface. Flexible NetFlow was set up to create the flows taking into account the same parameters as are used to create the flow by VBS. On the computer, we used *tcpreplay* to replay the PCAP files to the router with the maximal speed, which did not cause packet loss. At the same time, we used *nfacctd*, which is a part of PMACCT tools, to capture the Flexible NetFlow records sent by the router to the computer. The records, which contain the flow identifier (encoded as source MAC address) and the name of the application recognized by NBAR, were saved into text log files. This process is broadly elaborated in our technical report [15].

Table 2: Application Classes in the Dataset

Application	Number of Flows	Number of Megabytes
eDonkey	176581	2823.88
BitTorrent	62845	2621.37
FTP	876	3089.06
DNS	6600	1.74
NTP	27786	4.03
RDP	132907	13218.47
NETBIOS	9445	5.17
SSH	26219	91.80
Browser HTTP	46669	5757.32
Browser RTMP	427	3026.57
Unclassified	771667	5907.15

2.5 Dataset

Our dataset contains 1 262 022 flows captured during 66 days, between February 25, 2013 and May 1, 2013, which account for 35.69 GB of pure packet data. The application name tag was present for 520 993 flows (41.28 % of all the flows), which account for 32.33 GB (90.59 %) of the data volume. Additionally, 14 445 flows (1.14 % of all the flows), accounting for 0.28 GB (0.78 %) of data volume, could be identified based on the HTTP *content-type* field extracted from the packets. Therefore, we were able to successfully establish the ground truth for 535 438 flows (42.43 % of all the flows), accounting for 32.61 GB (91.37 %) of data volume. The remaining flows are unlabeled due to their short lifetime (below 1 s), which made VBS incapable to reliably establish the corresponding sockets. Only these successfully classified flows will be taken into account during the evaluation of the classifiers. However, all the flows are included in the publicly available traces. This ensures data integrity and the proper work of the classifiers, which may rely on coexistence of different flows. We isolated several application classes based on the information stored in the database (e.g., application labels, HTTP *content-type* field). The classes together with the number of flows and the data volume are shown in Table 2. We have published this labeled dataset with full packet payloads in [12]. Therefore, it can be used by the research community as a reference benchmark for the validation and comparison of network traffic classifiers.

3 Performance Comparison

This section provides a detailed insight into the classification results of different types of traffic by each of the classifiers. All these results are summarized in Table 3 and Table 4, where the ratio of correctly classified flows (i.e., precision or true positives), incorrectly classified flows (i.e., errors or false positives) and unclassified flows (i.e., unknowns) are respectively presented. The complete confusion matrix can be found in our technical

Table 3: Evaluation of DPI tools – Part 1

Application	Classifier	Correct [%]	Wrong [%]	Unclassified [%]
eDonkey	PACE	94.80	0.02	5.18
	OpenDPI	0.45	0.00	99.55
	L7-filter	34.21	13.70	52.09
	nDPI	0.45	6.72	92.83
	Libprotoident	98.39	0.00	1.60
	NBAR	0.38	10.81	88.81
BitTorrent	PACE	81.44	0.01	18.54
	OpenDPI	27.23	0.00	72.77
	L7-filter	42.17	8.78	49.05
	nDPI	56.00	0.43	43.58
	Libprotoident	77.24	0.06	22.71
	NBAR	27.44	1.49	71.07
FTP	PACE	95.92	0.00	4.08
	OpenDPI	96.15	0.00	3.85
	L7-filter	6.11	93.31	0.57
	nDPI	95.69	0.45	3.85
	Libprotoident	95.58	0.00	4.42
	NBAR	40.59	0.00	59.41
DNS	PACE	99.97	0.00	0.03
	OpenDPI	99.97	0.00	0.03
	L7-filter	98.95	0.13	0.92
	nDPI	99.88	0.09	0.03
	Libprotoident	99.97	0.00	0.04
	NBAR	99.97	0.02	0.02
NTP	PACE	100.00	0.00	0.00
	OpenDPI	100.00	0.00	0.00
	L7-filter	99.83	0.15	0.02
	nDPI	100.00	0.00	0.00
	Libprotoident	100.00	0.00	0.00
	NBAR	0.40	0.00	99.60

report [15].

Regarding the classification of P2P traffic, *eDonkey* is the first application studied. Only PACE, and especially Libprotoident, can properly classify it (precision over 94 %). nDPI and OpenDPI (that use the same pattern), as well as NBAR, can classify almost no eDonkey traffic (precision below 1 %). L7-filter classifies 1/3 of the flows, but it also produces many false positives by classifying more than 13 % of the flows as *Skype*, *NTP*, and *Finger*. The wrongly classified flows by nDPI were labeled as *Skype*, *RTP*, and *RTCP*, and by NBAR as *Skype*. The classification of *BitTorrent* traffic, the second P2P application studied, is not completely achieved by any of the classifiers. PACE and Libprotoident achieve again the highest precision (over 77 %). The rest of the classifiers present severe problems to identify this type of traffic. When misclassified, the BitTorrent traffic is usually classified as *Skype*.

The performance of most DPI tools with more traditional applications is significantly

Table 4: Evaluation of DPI tools – Part 2

Application	Classifier	Correct [%]	Wrong [%]	Unclassified [%]
	PACE	95.57	0.00	4.43
	OpenDPI	95.59	0.00	4.41
SSH	L7-filter	95.71	0.00	4.29
	nDPI	95.59	0.00	4.41
	Libprotoident	95.71	0.00	4.30
	NBAR	99.24	0.05	0.70
	PACE	99.04	0.02	0.94
	OpenDPI	99.07	0.02	0.91
RDP	L7-filter	0.00	91.21	8.79
	nDPI	99.05	0.08	0.87
	Libprotoident	98.83	0.16	1.01
	NBAR	0.00	0.66	99.34
	PACE	66.66	0.08	33.26
	OpenDPI	24.63	0.00	75.37
NETBIOS	L7-filter	0.00	8.45	91.55
	nDPI	100.00	0.00	0.00
	Libprotoident	0.00	5.03	94.97
	NBAR	100.00	0.00	0.00
	PACE	80.56	0.00	19.44
	OpenDPI	82.44	0.00	17.56
RTMP	L7-filter	0.00	24.12	75.88
	nDPI	78.92	8.90	12.18
	Libprotoident	77.28	0.47	22.25
	NBAR	0.23	0.23	99.53
	PACE	96.16	1.85	1.99
	OpenDPI	98.01	0.00	1.99
HTTP	L7-filter	4.31	95.67	0.02
	nDPI	99.18	0.76	0.06
	Libprotoident	98.66	0.00	1.34
	NBAR	99.58	0.00	0.42

higher. *FTP* traffic is usually correctly classified. Only L7-filter and NBAR present problems to label it. The false positives produced by L7-filter are because the traffic is classified as *SOCKS*. Table 3 also shows that all the classifiers can properly classify *DNS* traffic. Similar results are obtained for *NTP*, which almost all the classifiers can correctly classify it. However, NBAR completely miss the classification of this traffic. *SSH* was evaluated in its Linux version. Table 4 shows that NBAR almost classified all the flows while the rest of classifiers labeled more than 95 % of them.

Similar performance is also obtained with *RDP*, usually employed by VoIP applications, as shown in Table 4. Again, L7-filter and NBAR cannot classify this application at all. The false positives for L7-filter, Libprotoident, and NBAR are mainly due to *Skype*, *RTMP*, and *H323*, respectively.

Unlike previous applications, the results for *NETBIOS* are quite different. Surprisingly, NBAR and nDPI are the only classifiers that correctly label the NETBIOS

traffic. PACE can classify 2/3 of this traffic and OpenDPI only 1/4. On the other hand, the patterns from L7-filter and Libprotoident do not properly detect this traffic. The wrongly classified flows by Libprotoident are labeled as *RTP* and *Skype*, and by L7-filter as *eDonkey*, *NTP*, and *RTP*.

We also evaluated the *RTMP* traffic, a common protocol used by browsers and plugins for playing Flash content. It is important to note that only Libprotoident has a specific pattern for RTMP. Because of that, we have also counted as correct the RTMP traffic classified as *Flash* although that classification is not as precise as the one obtained by Libprotoident. L7-filter and NBAR cannot classify this type of traffic. The rest of the classifiers achieve similar precision, around 80 %. The surprising amount of false positives by nDPI is because some traffic is classified as *H323*. L7-filter errors are due to wrongly classified traffic as *Skype* and *TSP*.

Table 4 also presents the results regarding the *HTTP* protocol. All of them but L7-filter can properly classify most of the HTTP traffic. L7-filter labels all the traffic as *Finger* or *Skype*. nDPI classifies some HTTP traffic as *iMessage_Facetime*. The amount of errors from PACE is surprising, as this tool is usually characterized by very low false positive ratio. All the wrong classifications are labeled as *Meebo* traffic. The older *Meebo* pattern available in OpenDPI and the newer from nDPI seems not to have this problem.

Most incorrect classifications for all the tools are due to patterns that easily match random traffic. This problem especially affects L7-filter and, in particular, with the patterns used to match *Skype*, *Finger*, and *NTP* traffic. The deactivation of those patterns would considerably decrease the false positive ratio but it would disable the classification of those applications. In [4], the authors use a tailor-made configuration and post-processing of the L7-filter output in order to minimize this overmatching problem.

3.1 Sub-Classification of HTTP traffic

Our dataset also allows the study of HTTP traffic at different granularities (e.g., identify different services running over HTTP). However, only nDPI can sub-classify some applications at this granularity (e.g., *YouTube* or *Facebook*). Newer versions of PACE also provide this feature but we had no access to it for this study. Table 5 presents the results for four applications running over HTTP identified by nDPI. Unlike the rest of tools that basically classify this traffic as *HTTP*, nDPI can correctly give the specific label with precision higher than 97 %. Furthermore, the classification errors are caused by traffic that nDPI classifies as *HTTP* without providing the lower level label.

Another sub-classification that can be studied with our dataset is the *Flash* traffic over HTTP. However, the classification of this application is different for each tool making its comparison very difficult. PACE, OpenDPI, and nDPI have a specific pattern for this application. At the same time, these tools (as well as L7-filter) have specific patterns for video traffic, which may or may not run over HTTP. In addition, nDPI

Table 5: HTTP Sub-Classification by *nDPI*

Application	Correct [%]	Wrong [%]	Unclassified [%]
Google	97.28	2.72	0.00
Facebook	100.00	0.00	0.00
YouTube	98.65	0.45	0.90
Twitter	99.75	0.00	0.25

has specific labels for *Google*, *YouTube*, and *Facebook* that can also carry Flash traffic. Libprotoident and NBAR do not provide any pattern to classify Flash traffic over HTTP. Table 6 shows that nDPI can correctly classify 99.48 % of this traffic, 25.48 % of which is classified as *Google*, *YouTube*, or *Facebook*. PACE and OpenDPI can properly classify around 86 % of the traffic. The errors produced in the classification are almost always related to traffic classified as *HTTP* with the exception of L7-filter that classifies 86.49 % of the traffic as *Finger*.

Table 6: Flash Evaluation

Classifier	Correct [%]	Wrong [%]	Unclassified [%]
PACE	86.27	13.18	0.55
OpenDPI	86.34	13.15	0.51
L7-filter	0.07	99.67	0.26
nDPI	99.48	0.26	0.26
Libprotoident	0.00	98.07	1.93
NBAR	0.00	100.00	0.00

4 Discussion

This section extracts the outcomes from the results obtained during the performance comparison. Also, we discuss the limitations of our study. Table 7 presents the summary of the results from Section 3. The *Precision* (i.e., the first column) is computed similarly to Section 3, but we take into account all the applications together (i.e., 100 * # correctly classified flows / # total flows). However, this metric is dependent on the distribution of the dataset. Because of that, we also compute a second metric, the *Average Precision*. This statistic is independent from the distribution and is calculated as follow:

$$Avg.\ Precision = \frac{\sum_{i=1}^{N} \frac{correctly\ classified\ i\ flows}{total\ i\ flows}}{N}$$

where N is the number of applications studied (i.e., N = 10).

As it can be seen in Table 7, PACE is the best classifier. Even while we were not using the last version of the software, PACE was able to properly classify 94 % of our dataset. Surprisingly for us, Libprotoident achieves similar results, although this tool

Table 7: Summary

Classifier	Precision [%]	Average Precision [%]
PACE	94.22	91.01
OpenDPI	52.67	72.35
L7-filter	30.26	38.13
nDPI	57.91	82.48
Libprotoident	93.86	84.16
NBAR	21.79	46.72

only inspects the first four bytes of payload for each direction. On the other hand, L7-filter, and NBAR perform poorly in classifying the traffic from our dataset. The more fair metric, *Avg. Precision*, presents similar results. PACE is still the best classifier, however, it has increased the difference by several points to the second best classifier, Libprotoident. Unlike before, nDPI is almost as precise as Libprotoident with this metric. L7-filter and NBAR are still the tools that present the worst performance.

Nonetheless, the previous conclusions are obviously tied to our dataset. Although we have tried our best to emulate the real behavior of the users, many applications, behaviors and configurations are not represented on it. Because of that, it has some limitations. In our study we have evaluated 10 well-known applications, however adding more applications as *Skype* or *Spotify* is part of our ongoing future work. The results obtained from the different classifiers are directly related to those applications. Thus, the introduction of different applications could arise different outcomes. The traffic generated for building the dataset, although has been manually and realistically created, is artificial. The backbone traffic would carry different behaviors of the applications that are not fully represented in our dataset (e.g., P2P clients running on port 80). Therefore, the performance of the tools studied could not be directly extrapolated from the current results, but it gives an idea of their precision in the evaluated set of applications. At the same time, the artificially created traffic allowed us to publish the dataset with full packet payloads.

5 Conclusions

This paper presents the first step towards validating the reliability of the accuracy of the network traffic classifiers. We have compared the performance of six tools (i.e., PACE, OpenDPI, L7-filter, nDPI, Libprotoident, and NBAR), which are usually used for the traffic classification. The results obtained in Section 3 and further discussed in Section 4 show that PACE is, on our dataset, the most reliable solution for traffic classification. Among the open-source tools, nDPI and especially Libprotoident present the best results. On the other hand, NBAR and L7-filter present several inaccuracies that make them not recommendable as a ground-truth generator.

In order to make the study trustworthy, we have created a dataset using VBS [13].

This tool associates the name of the process to each flow making its labeling totally reliable. The dataset of more than 500 K flows contains traffic from popular applications like HTTP, eDonkey, BitTorrent, FTP, DNS, NTP, RDP, NETBIOS, SSH, and RDP. The total amount of data properly labeled is 32.61 GB. Furthermore, and more important, we release to the research community this dataset with full payload, so it can be used as a common reference for the comparison and validation of network traffic classifiers.

As the future work, we plan to extend this work by adding new applications to the dataset (e.g., Skype, Games) and especially focus on HTTP-based applications. We also plan to introduce new tools to the study (e.g., NBAR2).

References

[1] Alberto Dainotti, Antonio Pescapè, and Kimberly C. Claffy. Issues and future directions in traffic classification. *IEEE Network*, 26(1):35–40, 2012. DOI: 10.1109/MNET.2012.6135854.

[2] Silvio Valenti, Dario Rossi, Alberto Dainotti, Antonio Pescapè, Alessandro Finamore, and Marco Mellia. Reviewing Traffic Classification. In *Data Traffic Monitoring and Analysis*, pages 123–147. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-36784-7_6.

[3] Kensuke Fukuda. Difficulties of identifying application type in backbone traffic. In *2010 International Conference on Network and Service Management (CNSM)*, pages 358–361. IEEE, Niagara Falls, Ontario, Canada, October 2010. DOI: 10.1109/CNSM.2010.5691234.

[4] Valentín Carela-Español, Pere Barlet-Ros, Alberto Cabellos-Aparicio, and Josep Solé-Pareta. Analysis of the impact of sampling on NetFlow traffic classification. *Computer Networks*, 55:1083–1099, 2011. DOI: 10.1016/j.comnet.2010.11.002.

[5] Shane Alcock and Richard Nelson. Libprotoident: Traffic Classification Using Lightweight Packet Inspection. Technical report, University of Waikato, August 2012. Accessible: `http://www.wand.net.nz/publications/lpireport`.

[6] Francesco Gringoli, Luca Salgarelli, Maurizio Dusi, Niccolo Cascarano, Fulvio Risso, and Kimberly C. Claffy. Gt: picking up the truth from the ground for internet traffic. *ACM SIGCOMM Computer Communication Review*, 39(5):12–18, 2009. DOI: 10.1145/1629607.1629610.

[7] Alberto Dainotti, Francesco Gargiulo, Ludmila I. Kuncheva, Antonio Pescapè, and Carlo Sansone. Identification of traffic flows hiding behind TCP port 80. In *IEEE*

International Conference on Communications (ICC), pages 1–6. IEEE, Cape Town, South Africa, May 2010. DOI: 10.1109/ICC.2010.5502266.

[8] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, et al. Transport layer identification of P2P traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134. ACM New York, Taormina, Sicily, Italy, August 2004. DOI: 10.1145/1028788.1028804.

[9] Chaofan Shen and Leijun Huang. On detection accuracy of L7-filter and OpenDPI. In *2012 Third International Conference on Networking and Distributed Computing (ICNDC)*, pages 119–123. IEEE, Hangzhou, China, October 2012. DOI: 10.1109/ICNDC.2012.36.

[10] Alcock, Shane and Nelson, Richard. Measuring the Accuracy of Open-Source Payload-Based Traffic Classifiers Using Popular Internet Applications. In *IEEE Workshop on Network Measurements (WNM), the 38th IEEE Conference on Local Computer Networks (LCN)*. IEEE, Sydney, Australia, October 2013.

[11] Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Quantifying the accuracy of the ground truth associated with Internet traffic traces. *Computer Networks*, 55(5):1158–1167, April 2011. DOI: 10.1016/j.comnet.2010.11.006.

[12] Traffic classification at the Universitat Politècnica de Catalunya (UPC)., 2014. [Online]. Available: http://www.cba.upc.edu/monitoring/traffic-classification.

[13] Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for classification of traffic in computer networks. In *Proceedings of the 19th Telecommunications Forum TELFOR 2011*, pages 210–213. IEEE, Belgrade, Serbia, November 2011. DOI: 10.1109/TELFOR.2011.6143528.

[14] Volunteer-Based System for Research on the Internet, 2012. [Online]. Available: http://vbsi.sourceforge.net/.

[15] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification. Technical report, Department of Computer Architecture (DAC), Universitat Politècnica de Catalunya (UPC), June 2013. Accessible: https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2013,en.html.

Paper VIII

Multilevel Classification and Accounting of Traffic in Computer Networks

Tomasz Bujlow and Jens Myrup Pedersen

Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark
{tbu, jens}@es.aau.dk

This paper is submitted for review to *Computer Networks* – a journal from Elsevier.

Abstract

Existing tools for traffic classification are shown to be incapable of identifying the traffic in a consistent manner. For some flows only the application is identified, for others only the content, for yet others only the service provider. Furthermore, Deep Packet Inspection is characterized by extensive needs for resources and privacy or legal concerns. Techniques based on Machine Learning Algorithms require good quality training data, which are difficult to obtain. They usually cannot properly deal with other types of traffic, than they are trained to work with, and they are unable to detect the content carried by the flow, or the service provider. To overcome the drawbacks of already existing methods, we developed a

novel hybrid method to provide accurate identification of computer network traffic on six levels: *Ethernet*, *IP protocol*, *application*, *behavior*, *content*, and *service provider*. Our system built based on the method provides also traffic accounting and it was tested on 2 datasets. We have shown that our system gives a consistent, accurate output on all the levels. We also showed that the results provided by our system on the application level outperformed the results obtained from the most commonly used DPI tools.

Keywords

traffic classification, C5.0, Machine Learning Algorithms, computer networks, browser traffic, Internet traffic

1 Introduction

1.1 Background

Classification and accounting of traffic in computer networks is an important task. To ensure the proper quality for the users, Internet Service Providers (ISPs) are required to know how their networks are being used. To satisfy their needs, information about the network usage must be presented on multiple levels, which characterize different aspects of the traffic: layer 3, layer 4 and application protocols, the behavior, the carried content, and the service provider. At first, the knowledge of how particular applications contribute to the traffic volume allows to adjust the network structure and settings. For example, users, who prefer to download large amounts of data using Peer-to-Peer (P2P) applications, can be offered higher bandwidth during the night, while the Quality of Service (QoS) policies in the network can be adjusted to support the most commonly used interactive applications, such as Skype. However, each application can be used in many different ways. For example, HTTP clients, as web browsers, can be used to browse pages, stream Internet radios or Internet TVs, or download big files. Therefore, to assure the proper quality of delivery, the network providers require the information about how the applications are being used. ISPs can buy bandwidth from multiple providers, which are characterized by different pricing and links of different quality to various service providers (as Yahoo, Google, Facebook, etc). To reduce the cost and provide better quality to the users, it is important to know which services are most commonly used and what kind of content (audio, video, etc.) is being offered by the services.

The first challenge would be where and how to monitor the traffic. The best places for traffic monitoring are access, distribution, or core links of the ISPs networks, as we cannot expect to install any software on all of the users' devices. The examination of the traffic must not violate the law and users' privacy. Moreover, the measurements

must be done in real-time, or nearly real-time, to avoid storing huge amounts of data, which is not doable in high-speed infrastructures with links exceeding 10 Gbit/s, without involving large processing power and storage space.

1.2 Existing Methods for Traffic Classification

There are three main methods of traffic classification in computer networks: classification by transport-layer port numbers, Deep Packet Inspection (DPI), and statistical classification. A survey on various methods for traffic identification can be found in [1].

Distinguishing the traffic based on port numbers [2, 3] is a well-known method, which is implemented by many network devices, as routers and layer-3 switches. The method is fast, but it is not capable of detecting protocols which use dynamic port numbers, as Skype or other P2P [4–6]. The same concerns services, which use different port numbers than the default ones.

DPI is more flexible since it relies on inspection of the application payload [7]. Unfortunately, not all the applications leave patterns, which allow to precisely identify packets belonging to the applications. Therefore, some application protocols are not detected at all, overmatched (giving false positives), or undermatched (giving false negatives) – see the pattern descriptions of l-7 filter [8]. Classification methods, which use DPI, are slow and they require a lot of processing power, which makes them unfeasible for real-time processing in high-speed networks [4, 5]. Furthermore, it can be even impossible to use DPI if the payload is encrypted, the application signatures were changed [4], or the national law forbids to use this technique because of privacy and confidentiality concerns [4].

The statistical analysis, a newly emerged technology, is a reply to the drawbacks of the methods described above. There are two different techniques used in statistical identification of data: classification and clustering. Creating classification rules and data clusters based on statistical parameters by hand is slow and inefficient. For that purpose, tools based on Machine Learning Algorithms (MLAs) were invented, which are able to generate the rules or clusters based on sample data provided as the training input. The accuracy of statistical classification was assessed to be over 95 %, while the ability to distinguish various application protocols was comparable to DPI, and the speed was similar to port-based classification [2–4, 6, 7, 9–11]. However, current applications of MLAs have many drawbacks. At first, training of MLAs requires good quality data. Otherwise, we risk obtaining results of poor accuracy. At second, the classification attributes need to be carefully selected. If the classifier uses time-based attributes, the classification results can be biased by conditions in the network at the time of measurement. At third, if the classifier is trained to recognize only several selected types of traffic, its ability to classify traffic in the real network is also limited. In such case, any flow representing another type of traffic will be misclassified by being assigned to the most probable class of the included traffic types.

1.3 Our Contributions

All the classification methods described above have one thing in common: they are not able to identify the traffic on multiple levels in a consistent manner. The port-based classification usually gives the name of the application protocol, while its reliability is low. As we show later in Section 4.4, the currently available DPI tools are not suitable for this purpose as well. Either their results are a mix of application names, content names, and service provider names, while no consistency on any level is preserved (PACE, OpenDPI, nDPI, and Libprotoident), or the classification is given consistently only on the application level, while their accuracy is too low to use these tools for traffic accounting purposes (NBAR and L7-filter). Machine Learning based tools are not able to detect directly the content carried by the traffic or its service provider. However, consistent multi-level classification is needed to ensure proper accounting of the traffic in order to satisfy the broad requirements shown is Section 1.1 imposed by the network users, ISPs, and scientists. Because the output of the existing tools is not consistent (or not reliable), it is not easy to create a combined tool, which executes the particular classifiers and merges their output, so it becomes consistent on all the levels.

Therefore, we developed a hybrid classification method, which accurately identifies the traffic on multiple levels: *Ethernet*, *IP protocol*, *application*, *behavior*, *content*, and *service provider*. In fact, the identification of the *Ethernet*, *IP protocol* levels can be done accurately based on fixed fields in packet headers, so this contribution might seem to be trivial. However, these levels are not included in the output by the most common tools used for traffic classification (as DPI-based), although they are extremely important for traffic accounting purposes. Thus, based on the accounting performed by our system, the operators are able to see the number of flows and the traffic volume on the selected classification levels, which passed the monitored network interface during the selected time. The classification methods were designed for traffic accounting purposes and, therefore, they rely on some flow parameters, which makes the system incapable of a flow identification before the flow is terminated. It means that the system cannot be used for any security functions, as blocking the unwanted traffic from reaching the network.

Our classification methods give the ability to work also with the traffic, which is not defined to be identified by the system – the classification result is explicitly given as *UNKNOWN*, instead of assigning the flow to the most probable class, which is the most common behavior of similar tools based on Machine Learning techniques. In this paper, we show many different techniques (based on packet headers, C5.0 MLA, IP addresses) used to classify the traffic on multiple levels. The Machine Learning techniques use training data delivered by our host-based traffic monitoring tool, which ensures precision close to 100 % in data labeling. Furthermore, we present the full design, evaluation, and the practical open-source prototype of the system[1]. The prototype was

[1] Our Java implementation can be obtained for free from SourceForge [12]

built to demonstrate the methods shown in this paper and their accuracy, thus, it was not evaluated regarding the speed and processing requirements.

1.4 Structure of the Paper

The remainder of this paper is structured as follows. We start by introducing the classification and accounting methods in Section 2. The process of obtaining the supplementary inputs to the system is shown in Section 3. Our system was evaluated on two sets of data; the first including full packet payloads (Section 4) and the second only packet headers (Section 5). The results were discussed and compared with the results obtained by other classification tools in Section 6. Section 7 shows the related work in this area, while Section 8 concludes the paper.

2 Classification and Accounting Methods

Our system performs classification of Ethernet traffic independently on six levels:

1. Ethernet level (for example IP, IPX, APPLETALK)
2. IP protocol level (for example ICMP, TCP, UDP, EIGRP)
3. Application level (for example HTTP, SSH, BITTORRENT, EDONKEY)
4. Behavior level (for example WEBBROWSING, FILETRANSFER, STREAMING)
5. Content level (for example VIDEO, AUDIO)
6. Service provider level (for example YOUTUBE, MSN, WIKIPEDIA)

The traffic labeling is done by several modules, through which the traffic information is subsequently passed.

2.1 Traffic Capturing and Basic Classification Module

This module is responsible for capturing all the Ethernet frames from the network and for the identification on level 1 (*Ethernet*) and 2 (*IP protocol*). At first, the captured frames are inspected to obtain the value of the *EtherType* field, which determines the classification on level 1. The list of possible values for level 1 corresponds to the possible values of the *EtherType* field and can be found in the IEEE Public EtherType list [13]. Then, all the packets are grouped into flows in the following manner:

a) If the *EtherType* is IP, the network and transport layer headers are inspected as well. The flows are constructed based on the 5-tuple: local and remote IP addresses, local

and remote ports, and the transport protocol name. The level 2 class is determined based on the value of the *Type* field from the IP packet. The list of possible classes for level 2 can be found on the official IANA website [14]. If the class cannot be determined, it is set to *UNKNOWN*.

b) If the *EtherType* is of another type, the flows are constructed based on 3-tuple: local and remote MAC addresses, and the *EtherType* value. The level 2 class is always set to *UNKNOWN*.

When no new packet in the flow is noticed for over 1 minute, the flow is considered to be terminated and it is provided to the next module.

2.2 Application and Behavior Classification Module

This module is responsible for classification of the traffic on level 3 (*application*) and 4 (*behavior*). The classification is performed by C5.0, which is based on a machine learning algorithm. Only the traffic classified on level 2 as *TCP* or *UDP* needs to be examined by this module – the rest obtains the class *UNKNOWN* on both levels without any inspection. Both levels 3 and 4 are assessed together, since the behavior is directly associated with the particular application.

The list of possible values for levels 3 and 4 determined by this module is shown in Table 1. The values were selected to cover the most common types of traffic (regarding the number of flows and the number of transmitted bytes) collected by our user-based traffic analysis software and can be further extended when needed. The *UNKNOWN* behavior contains a mix of various application behaviors, which we are not able to separate and can also contain some elements, which belong to other, explicitly specified classes (as for example for *EDONKEY*, the *UNKNOWN* behavior level class can contain some elements of the *FILETRANSFER* class, which we were not able to separate). Note that the different behaviors can be evaluated only with respect to the particular application.

The classification by C5.0 relies on the following attributes:

- Local and remote port numbers.
- Transport protocol name.
- Number of packets and bytes carried by the flow.
- Percent of packets and bytes, which go in the inbound direction.
- Average, maximum, minimum, first quartile, median, third quartile, and standard deviation of total, inbound, and outbound packet size.
- Percent of total, inbound, and outbound packets, which carry any data.

Table 1: List of Applications and Their Behaviors Recognized by Our System

Application	Behavior
AMERICASARMY	UNKNOWN, GAMESESSION
BITTORRENT	UNKNOWN, FILETRANSFER
DHCP	INTERACTIVE
DNS	INTERACTIVE
DOWNLOADER	UNKNOWN, FILETRANSFER
EDONKEY	UNKNOWN, FILETRANSFER
FTP	UNKNOWN, CONTROL, FILETRANSFER
HTTP	UNKNOWN, FILETRANSFER, WEBBROWSING, WEBRADIO
HTTPS	UNKNOWN
NETBIOS	UNKNOWN
NTP	INTERACTIVE
RDP	UNKNOWN
RTMP	STREAMING
SKYPE	UNKNOWN, CONVERSATION
SSH	UNKNOWN
TELNET	INTERACTIVE
UNKNOWN	UNKNOWN

- Percent of total, small (< 70 B), and big (> 1320 B) packets carrying data, which are going into the inbound direction.
- Percent of total, inbound, and outbound packets carrying data, which are small (< 70 B) and big (> 1320 B).
- Percent of total, inbound, and outbound packets, which have ACK and PSH flag set.
- Percent of packets containing ACK, and PSH flag, which are going in the inbound direction.

We use mainly attributes based on packet sizes and to avoid using any time-related statistics, which are sensitive to disturbances in the network. The decision tree obtained from the training process allows to identify the application and behavior classes without the knowledge of anything besides the properties of the network and transport layers.

2.3 Content Classification Module

This module is responsible for classification of the traffic on level 5 (*content*). The classification relies either on static associations with the previous levels, or it is based on the IP addresses, where the module requires a file with mappings between the IP addresses and the types of the content (see Table 2). If this level cannot be determined, it is set to *UNKNOWN*.

Table 2: Mappings Between the Application & Behavior Classes and the Content

Application & Behavior	Content
HTTP FILETRANSFER	Based on IP addresses: AUDIO, VIDEO, UNKNOWN
HTTP WEBBROWSING	WWW
HTTP WEBRADIO	AUDIO
RTMP STREAMING	MULTIMEDIA
All the other classes	UNKNOWN

2.4 Service Provider Classification Module

This module is responsible for classification of the traffic on level 6 (*service provider*). At first, the class is tried to be identified based on the IP addresses. For that purpose, the classifier needs a list of IP addresses and the corresponding names of the service providers. Yet unclassified flows are classified as *P2P* if they directly connect two end-users (the application level was identified as *BITTORRENT*, *EDONKEY*, or *SKYPE*). Still unclassified flows are classified as *LOCAL* if the remote IP address belongs to the private IP addresses pools, or if the application level associated with that flows is *DHCP* or *NETBIOS*. All other flows are assumed to be *UNKNOWN*.

Afterwards, this module verifies if the classifications on lower levels are consistent with the detected service. In case of misfits (i.e., a flow classified as *BITTORRENT* was estimated to originate from YouTube based on its IP), the classifications on the *application*, *behavior*, and *content* levels are being changed to *UNKNOWN*.

2.5 Traffic Accounting Module

This module is responsible for accounting of the traffic. The information about the flows is directed to this module as the final step, after the flows are classified on all the levels. There are two modes available for logging: aggregated statistics and per-flow records. The following information is logged to create the per-flow records:

- Timestamp of the start of the flow.
- Local and remote MAC addresses (for non-IP flows only).
- Local and remote IP addresses (for IP flows only).
- Local and remote transport layer port numbers (for IP flows only).
- Number of packets and bytes in the flow, separately in both directions.
- Results from the classification on all the six levels.

The statistics can be aggregated by: the day of the measurement, local MAC address (for non-IP flows only), local IP address (for IP flows only), or the results from the

classification on all the six levels. For each aggregate, we log the number of flows, packets, and bytes, separately in both directions.

3 Obtaining the Supplementary Inputs

Three of our system modules require additional inputs:

a) *Application and behavior classification module* requires a file with decision trees, based on which the embedded C5.0 classifier can recognize the application and behavior associated with the particular flows.

b) *Content classification module* requires a file with mappings between the IP addresses and the most common types of content, which are from this IP address provided.

c) *Service provider classification module* requires a file with mappings between the IP addresses and the provider service names associated with them.

The process of obtaining the particular inputs is shown in Figure 1.

3.1 C5.0 Decision Trees

Creating the Ground-Truth

To create the ground-truth dataset, we used Volunteer-Based System (VBS) for Research on the Internet developed at Aalborg University [15]. The tool was released under The *GNU General Public License v3.0* and it is published as a SourceForge project [12]. VBS consists of clients installed on machines belonging to volunteers and of the server with the central database. The clients collect the information about the flows of all Internet traffic data to and from the machine (i.e., start time of the flow, number of packets contained by the flow, local and remote IP addresses, local and remote ports, transport layer protocol) together with detailed information about each packet (i.e., direction, size, TCP flags, and relative timestamp to the previous packet). For each flow, the associated process name obtained from the system sockets is also collected. Additionally, the system collects some information about the types of transferred HTTP contents, for example *text/css* or *video/x-mpg*. The captured information is transmitted to the VBS server, which stores the data in a MySQL database. Therefore, the dataset created based on the information from VBS can be used as a source of ground truth for training of the C5.0 classifier. The design of VBS shown in Figure 2 was initially described in [16] and the current version with further improvements and refinements is shown in [15].

However, there are some limitations of VBS, which have an impact on the collected dataset. The socket must be open for at least 1 second to be noticed by VBS, so the

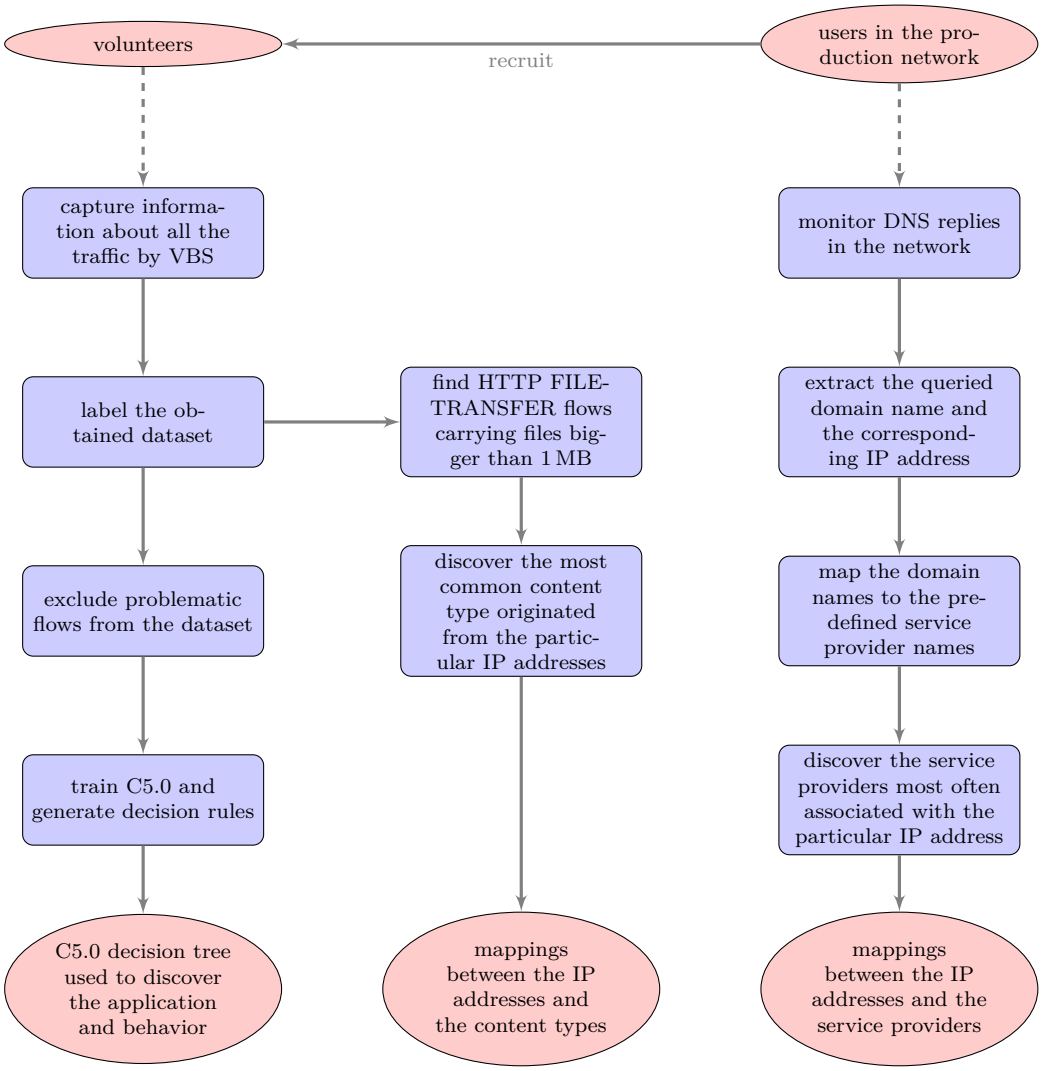

Figure 1: Training of the System Modules

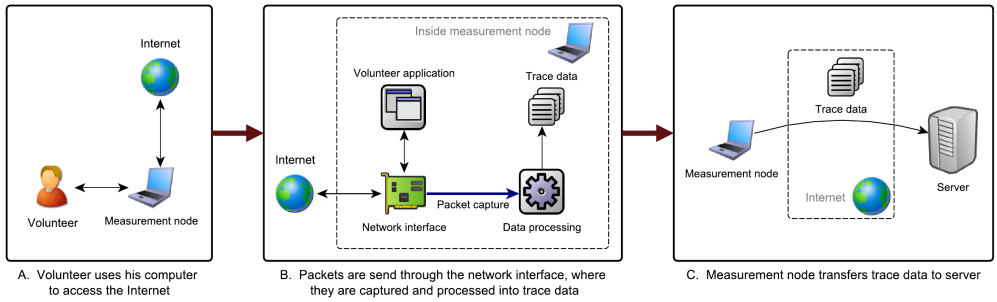

Figure 2: Overview of the VBS System [17]

process name could be collected. Therefore, in case of short flows (as 2-packets long DNS flows), the corresponding process name is not observed. Other limitations result from the host-monitoring nature of VBS, which collects only the data associated with the particular computer, on which it is installed. To create a representative dataset, the system must be installed on a representative group of users, who use their computers in a normal way.

Labeling the Ground-Truth

At first, we needed to create the rules to label the ground-truth dataset by *application* and *behavior* classes. All the rules were created by us manually, as it is not possible to automatically discover the character of the application based on the process name, or to discover the behavior of the traffic based on its general characteristics. The manual generation of classification rules is time-consuming, however, it allows us to make the classification consistent on all the levels. The rules are based mainly on our own observations regarding the traffic of the particular type. Another option would be to use an automatic training method, which could be based on results of DPI classification (as used in [18]), or on the process names from VBS. Unfortunately, by using output from DPI tools, we would mix the classifications on different levels. On the other hand, using the process names would create separate classes for each application (e.g., *uTorrent*, *BitTorrent*) instead of creating logical application groups. That would impose classification errors and add unneeded complexity, while the output would present lower value to the user. Additionally, there would be no way to discover the particular behavior of the application.

Labeling the flows with ground truth information on the *application* and *behavior* levels is split into two steps: we start by assigning the *application* class, and then, we assess the *behavior* within the particular application. The rules are processed from the most detailed to the most general, and only the previously unclassified cases are taken into account. For example, rules for *HTTP* traffic are processed before the rules

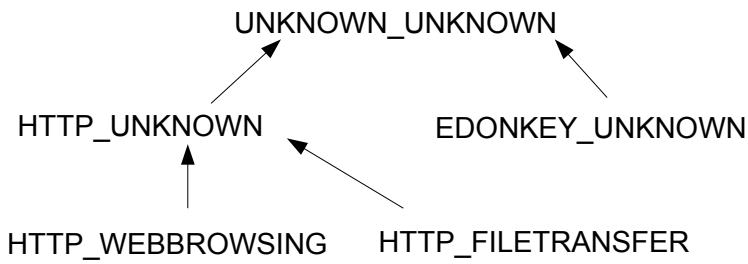

Figure 3: An example of Supersets and Subsets in Our Classification System

for the *BITTORRENT* traffic, because BitTorrent clients also use HTTP to download updates, etc. Obviously, this traffic must be classified as *HTTP*, not as *BITTORRENT*. The rules used to establish the ground truth on the *application* and *behavior* levels for all the flows in the VBS database are shown in our technical report [19].

Final Steps

Machine Learning based classification heavily depends on the quality of the training data, so the data used for training must be accurately assigned to the proper classes. In our case, the selected traffic classes are not disjoint but organized in supersets and subsets. For example, *HTTP WEBBROWSING* is a subset of *HTTP UNKNOWN*, which is in turn a subset of *UNKNOWN UNKNOWN* (see Figure 3). Our rules for labeling the ground-truth are not always able to assign a flow to a specific subset. Thus, some number of flows remains in one of the supersets.

As long as most of the flows are assigned to the proper subsets, Machine Learning based tools are able to use the data without decrease of accuracy. However, in some cases, the ground truth contains more elements of the particular class in the superset, instead of in the proper subset. That concerns short flows, as for example DNS flows, which are usually a few packets long, and for which we usually do not have captured the process names. Therefore, most of DNS flows remain in the *UNKNOWN UNKNOWN* class. If we use all the flows to generate the training data, we risk that some applications (as DNS) will not be classified at all. To avoid this issue, we need to exclude the problematic flows from the ground-truth dataset, so that the subsets always include more flows of the particular class than their supersets. Only the flows which do not have assigned application names and which do not contain any HTTP header are subject to be partly excluded. We decided that the number of flows contained by the superset and suspicious to belong to its subset must not exceed 25 % of the number of flows contained by the subset.

Finally, the training part of the groud-truth dataset was used to generate the decision

tree by C5.0.

3.2 Mappings between IP Addresses and Content Types

To obtain the required mappings, we again decided to use the data collected by VBS, as it also collects the value of the content-type field in HTTP headers. At first, we searched inside all the flows for files, which were at least 1 MB large and inspected their *content-type* field from the HTTP headers. If the file was of the type of *audio/xxx*, the source IP address of the file was assigned to the *AUDIO* class. If it was of the type of *video/xxx*, the source IP address was assigned to the *VIDEO* class. If it was of another type, the source IP address was assigned to the *UNKNOWN* class. After this step, we had records of IP addresses assigned to the *AUDIO*, *VIDEO*, or *UNKNOWN* classes – one record for each transmitted file. The final mappings were obtained by finding the most common classes for the particular IP addresses.

3.3 Mappings between IP Addresses and Service Providers

Our concept assumes creating a database of IP addresses assigned to the most often used services, based on DNS responses. Due to privacy issues, VBS does not inspect DNS packets, so processing of the DNS responses in order to obtain the IP addresses associated with the particular services was moved to another project, which relies on network-based monitoring. On a computer, which has access to the network traffic, we installed our software, which inspects all DNS replies and extracts from them the name of the service, which was queried, together with all the IP addresses associated with that service name. If the questioned domain name was an alias (CNAME), all the DNS replies are recurrently processed and all the IP addresses are extracted. DPI of DNS replies is fast and lightweight, so a dedicated host is able to process all the DNS replies even in a high-speed network.

It is worth mentioning that one machine, which possesses a single IP address, can be used by many different network services offered by different providers. The same IP address can be used by, for example, both Google and Yahoo!, or the questioned domain name can correspond to a proxy. It is not useful to process the DNS responses directly by the main classifier and to extract the IP addresses in the real time. A DNS response can be buffered on a host for many days, when the buffered information can be used to get the IP address of the particular domain name. In the meantime, the host can obtain multiple DNS responses with other domain names corresponding to the same IP address, so there is no possibility to accurately obtain the real domain name associated with the particular flow by monitoring the packets in the network. It would be possible only by host-based monitoring, when all requests to DNS cache will be handled as well.

The collected domain names, together with the IP addresses are stored in a table together with a counter, which informs us how many times the DNS responses contained the particular domain name associated with the particular IP address. Based on that,

we can see which domains were questioned the most, and we can create appropriate rules for the service providers (as for example for YouTube, the domain names must contain *youtube.*, *ytimg.com*, or *youtube-nocookie.*).

Based on the table, the mappings between the service provider names and the IP addresses can be made. Normally, each IP address is associated with the service provider name, that amounts for the highest number of occurrences for the given IP address. However, we can have multiple different lists with mappings, which will be used depending on the previous classifications of the flow (i.e., flows previously classified as *HTTP FILETRANSFER VIDEO* can use another list than flows classified as *RTMP STREAMING*, etc.).

4 Evaluation on a Dataset with Full Payloads

The evaluation of the multi-level classifier requires a few steps. The dataset used for the evaluation is created, the ground truth is established by labeling the dataset on all the levels, and the dataset is divided into the training and the testing part. The former part is used for creating the necessary input files (decision trees, mappings between the IP addresses and the content types or service providers), and the latter part is used for the evaluation of the classification accuracy.

4.1 Creating and Labeling the Dataset

To make any dataset useful for the purpose of the evaluation of our method, we needed to be able to label the traffic on all the levels. The dataset collected by VBS is able to deliver all the necessary information associated with each flow (as IP protocol, application name) but the name of the service provider. Therefore, we chose to use the dataset collected by a modified version of VBS, which we adjusted to collect also the packet payloads. It was used in collaboration with UPC BarcelonaTech to test the accuracy of different DPI tools (see the technical report published at UPC [20] and the conference paper [21]).

We started by establishing the ground truth – every flow from the dataset was classified on all the 6 levels using the objective information, including process names from the system sockets, and HTTP URLs from the HTTP headers.

Ethernet Level

All the flows were classified as *IP*, since VBS collects only IP traffic. However, this did not influence the obtained classification accuracy – the classification on that level is also based on the EtherType field, so the classification results would always be the same as the ground truth.

IP Protocol Level

The flows were classified as *TCP* or *UDP* based on the information extracted from the packet headers. VBS collects only TCP and UDP traffic. However, this did not influence the obtained classification accuracy – the classification on that level is also based on the IP protocol field, so the classification results would always be the same as the ground truth.

Application and Behavior Levels

To obtain the ground truth, we used the rules for generating the training data for the Machine-Learning based classifier on the application and behavior levels, which were processed in the same order as we described in Section 3.1. The rules were based on the process names from the system sockets and other objective information from VBS.

Content Level

To establish the ground truth, we made three steps. At first, we used the same static mappings as described in Section 2.3. Then, the class of *HTTP FILETRANSFER* flows (which is normally determined based on IP addresses), was determined based on the real content reflected by the *content-type* field from HTTP headers.

Service Provider Level

To establish the ground truth, we made the steps in the same order as described in Section 2.4. Although we had all the DNS packets together with their payload, we could not use them to generate the ground truth, as one IP address could correspond to many different service providers. Therefore, instead of using the DNS mappings, we classified the flows based on the URLs found in the HTTP header of the packets belonging to these flows. The limitation of this method is that we were able to classify only HTTP flows. Based on the collected data, we chose 14 service providers to use in the experiment. As the second step, we used the static mappings as described in Section 2.4.

4.2 Dataset

Based on the established ground truth, we show the statistics about the applications and their behaviors contained by our dataset. The dataset consisted of 1 262 022 flows (303 189 TCP and 958 833 UDP) amounting for 35.69 GB (33.97 GB TCP and 1.72 GB UDP).

Table 3: Ground-Truth Classification of Our Dataset with Full Payloads

Ethernet	IP	Application	Behavior	Content	Service Provider	Flows	MB	Flows [%]	MB [%]
IP	TCP	BITTORRENT	FILETRANSFER	UNKNOWN	P2P	572	1046.94	0.05	2.86
IP	TCP	BITTORRENT	UNKNOWN	UNKNOWN	P2P	29967	75.94	2.37	0.21
IP	TCP	DNS	INTERACTIVE	UNKNOWN	UNKNOWN	2	0	0	0
IP	TCP	EDONKEY	FILETRANSFER	UNKNOWN	P2P	135	2755.48	0.01	7.54
IP	TCP	EDONKEY	UNKNOWN	UNKNOWN	P2P	1423	5.3	0.11	0.01
IP	TCP	FTP	CONTROL	UNKNOWN	UNKNOWN	61	0.58	0	0
IP	TCP	FTP	FILETRANSFER	UNKNOWN	UNKNOWN	797	3088.42	0.06	8.45
IP	TCP	FTP	UNKNOWN	UNKNOWN	UNKNOWN	18	0.06	0	0
IP	TCP	HTTP	FILETRANSFER	UNKNOWN	X	144	1631.6	0.01	4.46
IP	TCP	HTTP	FILETRANSFER	VIDEO	X	295	2160.21	0.02	5.91
IP	TCP	HTTP	UNKNOWN	UNKNOWN	X	15840	443.92	1.25	1.21
IP	TCP	HTTP	WEBBROWSING	WWW	X	45934	2248.9	3.63	6.15
IP	TCP	HTTPS	UNKNOWN	UNKNOWN	UNKNOWN	8539	833.85	0.68	2.28
IP	TCP	RDP	UNKNOWN	UNKNOWN	UNKNOWN	132907	13218.47	10.53	36.16
IP	TCP	RMTP	STREAMING	MULTIMEDIA	UNKNOWN	145	3209.49	0.01	8.78
IP	TCP	SSH	UNKNOWN	UNKNOWN	UNKNOWN	26219	91.8	2.08	0.25
IP	TCP	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	40191	3978.46	3.18	10.88
IP	UDP	BITTORRENT	FILETRANSFER	UNKNOWN	P2P	466	1433.98	0.04	3.92
IP	UDP	BITTORRENT	UNKNOWN	UNKNOWN	P2P	31840	64.51	2.52	0.18
IP	UDP	DNS	INTERACTIVE	UNKNOWN	UNKNOWN	6598	1.74	0.52	0
IP	UDP	EDONKEY	UNKNOWN	UNKNOWN	P2P	175023	63.1	13.87	0.17
IP	UDP	NETBIOS	UNKNOWN	UNKNOWN	LOCAL	9445	5.17	0.75	0.01
IP	UDP	NTP	INTERACTIVE	UNKNOWN	UNKNOWN	27786	4.03	2.2	0.01
IP	UDP	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	707675	188.95	56.07	0.52

Table 4: Ground-Truth Classification of Service Providers for the HTTP Traffic

Service	Flows	MB	Flows [%]	MB [%]
YOUTUBE	2210	1949.19	3.55	30.06
UNKNOWN	22182	1349.45	35.65	20.81
ORACLE	103	841.74	0.17	12.98
FACEBOOK	5285	681.33	8.50	10.51
WIKIPEDIA	4078	508.19	6.55	7.84
YAHOO	16913	487.49	27.19	7.52
GOOGLE	3273	326.88	5.26	5.04
JUSTIN.TV	4075	134.05	6.55	2.07
UBUNTU.COM	247	102.65	0.40	1.58
MICROSOFT.COM	1301	55.56	2.09	0.86
DOUBLECLICK.NET	2072	38.72	3.33	0.60
TWITTER	297	7.59	0.48	0.12
TRIBALFUSION.COM	95	1.59	0.15	0.02
SCORECARDRESEARCH.COM	82	0.20	0.13	0.00

We were not able to establish the ground truth on the application level for 59.25 % of flows, which amounted for 11.40 % of the total data volume. TCP flows amounted for 3.18 % of the total number of flows and 10.88 % of the total data volume without established ground truth. UDP flows amounted for 56.07 % of the total number of flows and 0.52 % of the total data volume without established ground truth. In fact, this is caused by a huge number of unclassified DNS flows, which are usually 2-packets long, so the corresponding socket application name could not be observed during the capture. The overall ground-truth classification of our dataset on all the levels is shown in Table 3.

For some HTTP flows, we detected the service provider based on the *URL* field in the HTTP header. The statistics for HTTP flows are shown in Table 4 – we labeled 64.35 % of HTTP flows amounting for 79.19 % of the HTTP data volume.

In total, for all the flows, we detected the service provider for 22.89 % of flows amounting for 28.97 % of the total data volume.

We split our dataset into 2 disjoint sets, where each of them contained approximately the same number of flows. Each flow was randomly assigned to either set 1 or set 2. Both sets were used during the training and testing of the classifier. At first, set 1 was used as the training set and test 2 as the test set, then the same procedure was done with both sets swapped. Thanks to that, we were able to test the classifier on all the flows contained by our dataset, while at the same time no flow was present both in the training and in the test set.

4.3 Results

The classification results are shown and discussed separately for each classification module. As the results are calculated on the per-class basis, they are only shown in terms of

flows – the classes represent the same types of flows, so the percents of flows and bytes and similar.

There are two most common metrics provided by the scientific publications for the results concerning classification techniques: precision and sensitivity (called also completeness or recall). Then, the final accuracy is calculated as a weighted mean from these two basic metrics. Our analysis method is based on only one metric, this is *sensitivity* – we take into account the original classes of flows and then we look if the flows were assigned to the proper traffic classes or not. There are several grounds for that. At first, we have a lot of unknown flows in the test set. The unknown traffic can be classified either as unknown, or it can be recognized by the classifier as a part of another traffic class. In this case, we assume that the classification is correct and that the classifier recognized properly the class of the traffic, which was initially not marked in the ground-truth set. In fact (as we compared the results with PACE), the classifier correctly recognized most of this traffic, so the test set was classified better (much less unknown cases) by our classifier than the initial ground-truth was. However, as we cannot prove that the unknown traffic was classified correctly, we decided to avoid using the per-class *precision*. As all the traffic in our set was assigned to a class (or left as unknown, if we were not able to determine the class), all the classification error are always visible as the lower class *sensitivity*.

Traffic Capturing and Basic Classification Module

As we expected, this module reached 0.00 % of error rate and classified properly all the cases. All 1 262 022 flows were classified on the *Ethernet* level as *IP*. On the *IP protocol* level, 303 189 (24.02 %) of the flows were classified as TCP and 958 833 (75.98 %) as UDP.

Application and Behavior Classification Module

The results obtained by this module are shown in Table 5. We present the number of flows in each traffic category (based on the ground truth) together with the percentage of flows, which were classified correctly, wrong, or as *UNKNOWN*. There are two categories for the flows correctly classified. The first category, *correct*, means that the flows were assigned to the same class as the original one or to a subset of the original class. For example, flows from the *BITTORRENT UNKNOWN* class were classified as *BITTORRENT UNKNOWN*, or *BITTORRENT FILETRANSFER*. The second category, *correct-lg*, means that we obtained results of lower granularity than the original class, but still both application and behavior levels are not misclassified. For example, flows from the *BITTORRENT FILETRANSFER* class were classified as *BITTORRENT UNKNOWN*.

As observed, the total error rate for all the flows was 0.08 %, while 0.54 % of flows remained unknown. If we look at the average from all the traffic classes, the error

Table 5: Classification by the *Application and Behavior Classification Module* for the Particular Types of the Traffic

Application & Behavior	Flows	Correct [%]	Correct-LG [%]	Wrong [%]	Unknown [%]
BITTORRENT FILETRANSFER	1038	97.59	0.39	1.16	1.06
BITTORRENT UNKNOWN	61807	98.17	0.00	0.22	1.61
DNS INTERACTIVE	6600	95.00	0.00	0.03	4.97
EDONKEY FILETRANSFER	135	87.41	2.22	4.44	5.93
EDONKEY UNKNOWN	176446	99.52	0.00	0.08	0.40
FTP CONTROL	61	98.36	0.00	0.00	1.64
FTP FILETRANSFER	797	97.24	0.00	1.00	1.76
FTP UNKNOWN	18	33.33	0.00	0.00	66.67
HTTP FILETRANSFER	439	92.94	4.78	1.82	0.46
HTTP UNKNOWN	15840	99.72	0.00	0.21	0.06
HTTP WEBBROWSING	45934	87.38	12.47	0.10	0.05
HTTPS UNKNOWN	8539	93.64	0.00	0.20	6.16
NETBIOS UNKNOWN	9445	100.00	0.00	0.00	0.00
NTP INTERACTIVE	27786	100.00	0.00	0.00	0.00
RDP UNKNOWN	132907	100.00	0.00	0.00	0.00
RTMP STREAMING	145	88.97	0.00	2.76	8.28
SSH UNKNOWN	26219	99.49	0.00	0.02	0.50
For all flows together	514156	98.26	1.12	0.08	0.54
Average from all classes		92.28	1.17	0.71	5.86

rate was slightly bigger (0.71 %) and 5.86 % of flows remained unknown. Additionally, we tried to identify 747 866 flows, which ground truth was given as *UNKNOWN UNKNOWN*: 88.49 % of them were classified as *DNS INTERACTIVE*, 4.20 % as *EDONKEY UNKNOWN*, 1.13 % as other classes, while only 6.18 % remained as *UNKNOWN UNKNOWN*. It means that the classifier built by us was able to recognize almost all the traffic for which we were not able to establish the ground truth due to the unknown application name.

Content Classification Module

The results of the classification of the particular types of contents is shown in Table 6. As presented, we did not obtain any false classification within the flows from the *MULTIMEDIA*, *VIDEO*, and *WWW* classes. The rate of the unclassified flows is heavily influenced by the size of the training and testing sets used in the experiment. In our case, the ground truth for the flows labeled as *UNKNOWN* was established as non-*MULTIMEDIA*, non-*VIDEO*, and non-*WWW*. We obtained 0.22 % of errors for this class, which means that the same IP addresses were used both for the flows transmitting files from the *MULTIMEDIA*, *VIDEO*, and *WWW* classes, and for flows transmitting other kinds of files. In total, we classified properly 99.32 % of flows, 0.22 % of flows were classified wrong, and 0.47 % remained unknown.

Table 6: Classification by the *Content Classification Module*

Content	Flows	Correct [%]	Wrong [%]	Unknown [%]
MULTIMEDIA	145	88.97	0.00	11.03
VIDEO	295	77.29	0.00	22.71
WWW	45934	87.38	0.00	12.62
UNKNOWN	1215648	99.78	0.22	0.00
All flows	1262022	99.32	0.22	0.47
Average from classes		88.36	0.06	11.59

Table 7: Classification by the *Service Provider Classification Module*

Service provider	Flows	Correct [%]	Wrong [%]	Unknown [%]
DOUBLECLICK.NET	2072	70.70	26.59	2.70
FACEBOOK	5285	97.35	1.06	1.59
GOOGLE	3273	41.34	42.65	16.01
JUSTIN.TV	4075	50.38	8.20	41.42
LOCAL	9445	100.00	0.00	0.00
MICROSOFT.COM	1301	84.24	15.14	0.61
ORACLE	103	65.05	19.42	15.53
P2P	239426	99.25	0.01	0.74
SCORECARDRESEARCH	82	1.22	95.12	3.66
TRIBALFUSION.COM	95	85.26	0.00	14.74
TWITTER	297	85.19	7.07	7.74
UBUNTU.COM	247	99.19	0.00	0.81
WIKIPEDIA	4078	98.58	0.81	0.61
YAHOO	16913	91.12	4.88	4.00
YOUTUBE	2210	82.67	15.02	2.31
All flows	288902	96.95	1.34	1.71
Average from classes		76.77	15.73	7.50

Service Provider Classification Module

The results of classification of particular service providers is shown in Table 7. The correct service provider name was given to 96.95 % of classified flows, the wrong service provider name was given to 1.34 % of flows, while for 1.71 % it remained unknown. A significant percent of errors originated from *DOUBLECLICK.NET* and *SCORECARDRESEARCH.COM*, which represent advertising services. It looks like their physical infrastructure is not centralized, but distributed among servers of other services, to which they supply advertisements. We also observed a significant misclassification between *GOOGLE* and *YOUTUBE*, as it seems that they use the same IP addresses.

4.4 Comparison with the Results from DPI Tools on the Application Level

The dataset containing packets with full payloads was used previously in [20] and [21] to evaluate the accuracy of various DPI traffic classifiers: PACE, OpenDPI, nDPI, Libpro-

Table 8: DPI Tools Included in Our Comparison

Name	Version	Released	Identified Applications
PACE	1.41	June 2012	1000
OpenDPI	1.3.0	June 2011	100
nDPI	rev. 6391	March 2013	170
L7-filter	2009.05.28	May 2009	110
Libprotoident	2.0.6	Nov 2012	250
NBAR	15.2(4)M2	Nov 2012	85

toident, NBAR, and two different variants of L7-filter. Therefore, all the flows contained by the dataset were labeled by all of the DPI tools, which allowed us to compare their accuracy with the accuracy of our system. Table 8 shows the versions of the DPI tools used in the experiment together with the number of applications recognized by them. L7-filter was tested in two different configurations. In the first version (*L7-filter-all*), we activated all the patterns, giving a low priority to the patterns marked as *overmatching*. In the second version (*L7-filter-sel*), the patterns declared as *overmatching* were not activated. In both cases, the patterns are matched to the first 10 packets or the first 10 000 B in each traffic direction.

All the tools provide results on different levels of granularity. For example, Libprotoident provides only transport protocol name if it is not able to detect the application. On the other hand, PACE, OpenDPI and nDPI sometimes provide only the content-level classification (as *FLASH*) without indicating what the application really is (for *FLASH* it can be *HTTP*, *RTMP*, etc). Because most of the results obtained from these DPI tools are on the *application* level, we decided to compare the classification results on our *application* level. The results provided by the DPI tools were considered to be correct only if the proper result on the *application* level was returned. In the comparison, we included only these cases, for which the ground truth was obtained. The percent of correct classification by all the tools is shown in Table 9. The percents concern the class *sensitivity*. As the classification levels obtained from the DPI tools are not consistent, we are not able to map the flows 1:1 to a particular application or protocol – some flows should be mapped to many groups at once, for example, to HTTP and BitTorrent (if it is the BitTorrent application which uses HTTP to download a tracker file from a website). Therefore, it is hard to decide how the precision is affected itself. However, it is generally easy to calculate the sensitivity: we take into account all the BitTorrent flows and determine how they are classified – regardless if the result is BitTorrent or HTTP, both the results are recognized as correct.

Table 9: Comparison of % Correct Classifications for the Particular Applications by Our System and Various DPI Tools

Application	Flows	Multi-level	PACE	Open-DPI	nDPI	Libproto-ident	NBAR	L7-Filter-All	L7-Filter-Sel
BITTORRENT	62845	98.14	81.40	27.09	55.92	77.19	27.30	42.22	42.24
DNS	6600	95.00	99.97	99.97	99.88	99.97	99.97	98.95	98.95
EDONKEY	176581	99.51	94.80	0.45	0.45	98.40	0.38	34.22	0.00
FTP	876	95.32	96.00	95.32	95.32	94.63	39.27	5.14	5.14
HTTP	62213	99.83	92.50	94.00	73.89	98.88	99.35	4.41	27.84
HTTPS	8539	93.64	91.73	93.76	39.67	92.65	85.62	72.99	90.06
NETBIOS	9445	100.00	66.66	24.63	100.00	0.00	100.00	0.00	0.00
NTP	27786	100.00	100.00	100.00	100.00	100.00	0.40	99.83	0.00
RDP	132907	100.00	99.06	99.09	99.07	98.85	0.00	0.00	0.00
RTMP	145	88.97	0.00	0.00	0.00	87.59	0.00	0.00	0.00
SSH	26219	99.49	95.57	95.59	95.59	95.71	99.24	95.71	95.71
For all flows together	514156	99.38	93.78	54.19	55.76	94.04	25.18	30.21	16.19
Average from all classes		97.44	84.29	65.34	67.96	86.49	48.06	40.31	31.34

The comparison shows that our multilevel classification system provided very precise results. The overall accuracy of our system was 99.38 % and the average per-class accuracy was 97.44 %. The values are higher than the accuracy of the tested DPI tools.

5 Evaluation on the Full VBS Dataset

After the small-scale system evaluation on the dataset containing full packet payloads, we decided to repeat the experiment on a much bigger dataset. For this purpose, we used the full dataset collected by VBS and stored at Aalborg University. The dataset contained traces collected by VBS clients installed on 54 machines belonging to volunteers in Denmark and Poland during around 1.5 years from December 27, 2011 to June 12, 2013. The volunteers were both private and corporate users. Such kind of setup ensures high diversity of the data despite the relatively limited number of installations. VBS in its original version did not collect packet payloads, so we were able to test the classification only on the *application*, *behavior*, and *content* levels. However, the classification results on the *Ethernet* and *IP provider* levels are accurate by nature, so no testing was needed. The methods of obtaining the ground truth and testing the classifier were the same as for the dataset with full packet payloads, which was described previously in Section 4.1.

5.1 Dataset

Based on the established ground truth, we show the statistics about the applications and their behaviors contained by our dataset. The dataset consisted of 23 333 721 flows (11 283 867 TCP and 12 049 854 UDP) amounting for 1 677.96 GB (1 114.85 GB TCP and 563.11 GB UDP).

We were not able to establish the ground truth on the *application* level for 26.33 % of flows, which amounted for 15.33 % of the total data volume. TCP flows amounted for 13.08 % of the total number of flows and 10.52 % of the total data volume without established ground truth. UDP flows amounted for 13.25 % of the total number of flows and 4.81 % of the total data volume without established ground truth. In fact, this was caused by a huge number of unclassified DNS flows, which are usually 2-packets long, so the corresponding socket application name could not be observed during the capture. The overall ground-truth classification of our VBS dataset is shown in Table 10.

Table 10: Ground-Truth Classification of Our VBS Dataset

IP Protocol	Application	Behavior	Content	Flows	MB	Flows [%]	MB [%]
TCP	BITTORRENT	FILETRANSFER	UNKNOWN	32774	154287.09	0.14	8.98
TCP	BITTORRENT	UNKNOWN	UNKNOWN	3235831	35263.55	13.87	2.05
TCP	DNS	INTERACTIVE	UNKNOWN	5	0.01	0.00	0.00
TCP	DOWNLOADER	FILETRANSFER	UNKNOWN	284	20499.20	0.00	1.19
TCP	DOWNLOADER	UNKNOWN	UNKNOWN	122503	3421.99	0.53	0.20
TCP	EDONKEY	FILETRANSFER	UNKNOWN	2719	71881.22	0.01	4.18
TCP	EDONKEY	UNKNOWN	UNKNOWN	22700	1809.43	0.10	0.11
TCP	FTP	CONTROL	UNKNOWN	105	0.35	0.00	0.00
TCP	FTP	FILETRANSFER	UNKNOWN	124	11713.73	0.00	0.68
TCP	FTP	UNKNOWN	UNKNOWN	11	0.06	0.00	0.00
TCP	HTTP	FILETRANSFER	AUDIO	1439	9313.63	0.01	0.54
TCP	HTTP	FILETRANSFER	UNKNOWN	9726	212182.14	0.04	12.35
TCP	HTTP	FILETRANSFER	VIDEO	23511	189091.28	0.10	11.01
TCP	HTTP	UNKNOWN	UNKNOWN	1583731	50895.30	6.79	2.96
TCP	HTTP	WEBBROWSING	WWW	2355298	72690.14	10.09	4.23
TCP	HTTP	WEBRADIO	AUDIO	326	19012.81	0.00	1.11
TCP	HTTPS	UNKNOWN	UNKNOWN	748695	37507.37	3.21	2.18
TCP	RDP	UNKNOWN	UNKNOWN	12	0.01	0.00	0.00
TCP	RTMP	STREAMING	MULTIMEDIA	497	67996.21	0.00	3.96
TCP	SKYPE	CONVERSATION	UNKNOWN	1588	2925.00	0.01	0.17
TCP	SKYPE	UNKNOWN	UNKNOWN	89557	346.54	0.38	0.02
TCP	UNKNOWN	UNKNOWN	UNKNOWN	3052431	180764.30	13.08	10.52
UDP	AMERICASARMY	GAMESESSION	UNKNOWN	80	111.36	0.00	0.01
UDP	AMERICASARMY	UNKNOWN	UNKNOWN	6260	1.69	0.03	0.00
UDP	BITTORRENT	FILETRANSFER	UNKNOWN	52931	391440.00	0.23	22.78
UDP	BITTORRENT	UNKNOWN	UNKNOWN	8200616	65932.23	35.14	3.84
UDP	DHCP	INTERACTIVE	UNKNOWN	4459	2.82	0.02	0.00
UDP	DNS	INTERACTIVE	UNKNOWN	158314	47.07	0.68	0.00
UDP	DOWNLOADER	UNKNOWN	UNKNOWN	9872	18.44	0.04	0.00
UDP	EDONKEY	UNKNOWN	UNKNOWN	59896	19.18	0.26	0.00
UDP	NETBIOS	UNKNOWN	UNKNOWN	13133	6.45	0.06	0.00
UDP	NTP	INTERACTIVE	UNKNOWN	58375	8.57	0.25	0.00
UDP	SKYPE	CONVERSATION	UNKNOWN	892	36256.00	0.00	2.11
UDP	SKYPE	UNKNOWN	UNKNOWN	394005	163.82	1.69	0.01
UDP	UNKNOWN	UNKNOWN	UNKNOWN	3091021	82611.96	13.25	4.81

5.2 Results

The classification results are shown and discussed separately for each classification module. As the results are calculated on the per-class basis, they are only shown in terms of flows – the classes represent the same types of flows, so the percents of flows and bytes and similar. The *error* metric is calculated in the same manner as described previously in Section 4.3.

Application and Behavior Classification Module

The results obtained by this module are shown in Table 11. We present the number of flows in each traffic category (based on the ground truth) together with the percentage of flows, which were classified correctly, wrong, or as *UNKNOWN*. The category notation is the same as for the dataset with full packet payloads, which was previously described in Section 4.3.

As observed, the total error rate for all the flows was 0.09 %, while 0.75 % of flows remained unknown. If we look at the average from all the traffic classes, the error rate was bigger (6.03 %) and 6.31 % of flows remained unknown. As shown in Table 11, the per-class error rate was higher due to several classes, which contained only a few flows. An example can be *RDP UNKNOWN*, which contained only 12 flows, which were mostly misclassified. However, the misclassification was not only made due to poor training on a limited number of flows; in our dataset we could see that the possessed by us *RDP UNKNOWN* flows were just connection requests, which were probably rejected from the server, so no real RDP connection was established. Additionally, we tried to identify 6 143 452 flows, which ground truth was given as *UNKNOWN UNKNOWN*: 29.41 % of them were classified as *DNS INTERACTIVE*, 2.42 % as *BITTORRENT UNKNOWN*, 1.97 % as other classes, while 66.20 % remained as *UNKNOWN UNKNOWN*.

Content Classification Module

The results of the classification of the particular types of contents is shown in Table 12. As presented, we obtained very low classification error (below 1 %) within the flows from the *MULTIMEDIA*, *VIDEO*, and *WWW* classes, and higher classification error of *AUDIO* flows (6.01 %). The rate of the unclassified flows fluctuated around 13 % for all the classes. In our case, the ground truth for the flows labeled as *UNKNOWN* was established as non-*AUDIO*, non-*MULTIMEDIA*, non-*VIDEO*, and non-*WWW*. We obtained only 1.06 % of errors for this class, which means that the same IP addresses were used both for the flows transmitting files from the *AUDIO*, *MULTIMEDIA*, *VIDEO*, and *WWW* classes, and for flows transmitting other kinds of files. In total, we classified properly 97.62 % of flows, 0.96 % of flows were classified wrong, and 1.42 % remained unknown.

Table 11: Classification by the *Application and Behavior Classification Module* for the Particular Types of the Traffic in VBS Dataset

Application & Behavior	Flows	Correct [%]	Correct-LG [%]	Wrong [%]	Unknown [%]
AMERICASARMY GAMESESSION	80	78.75	0.00	15.00	6.25
AMERICASARMY UNKNOWN	6260	99.89	0.00	0.08	0.03
BITTORRENT FILETRANSFER	85705	99.47	0.11	0.12	0.30
BITTORRENT UNKNOWN	11436447	99.78	0.00	0.02	0.20
DHCP INTERACTIVE	4459	99.57	0.00	0.00	0.43
DNS INTERACTIVE	158319	91.05	0.00	0.08	8.88
DOWNLOADER FILETRANSFER	284	70.42	8.10	15.14	6.34
DOWNLOADER UNKNOWN	132375	96.13	0.00	2.06	1.81
EDONKEY FILETRANSFER	2719	92.42	0.88	4.74	1.95
EDONKEY UNKNOWN	82596	96.53	0.00	1.99	1.48
FTP CONTROL	105	79.05	0.00	0.00	20.95
FTP FILETRANSFER	124	71.77	0.00	4.03	24.19
FTP UNKNOWN	11	54.55	0.00	0.00	45.45
HTTP FILETRANSFER	34676	97.56	1.52	0.80	0.12
HTTP UNKNOWN	1583731	99.13	0.00	0.01	0.86
HTTP WEBBROWSING	2355298	86.06	13.30	0.03	0.61
HTTP WEBRADIO	326	92.64	0.31	7.06	0.00
HTTPS UNKNOWN	748695	91.94	0.00	0.41	7.66
NETBIOS UNKNOWN	13133	99.97	0.00	0.00	0.03
NTP INTERACTIVE	58375	99.80	0.00	0.01	0.19
RDP UNKNOWN	12	8.33	0.00	83.33	8.33
RTMP STREAMING	497	85.92	0.00	4.83	9.26
SKYPE CONVERSATION	2480	89.35	1.09	3.95	5.60
SKYPE UNKNOWN	483562	98.51	0.00	0.95	0.55
For all flows together	17190269	97.33	1.83	0.09	0.75
Average from all classes		86.61	1.05	6.03	6.31

6 Discussion and Comparison of the Results

The results show that our classifier is able to identify the traffic on multiple different levels with high accuracy. What is unique for tools relying on Machine Learning capabilities, our classifier can properly handle traffic from non-recognized applications, marking it as *UNKNOWN*. We also compared the results of classification by our classifier to the results given by various DPI tools.

PACE. This DPI tool from *ipoque* [22] provides very accurate results on the *application* level. We assessed its accuracy as 93.78 % on our dataset with full payloads. However, some of the results are given on *content* level instead, as *FLASH*, *QUICKTIME*, or *WINDOWSMEDIA*. In these cases, we do not really know what is the application. For example, *FLASH* content can be transmitted by both *HTTP* or *RTMP* application protocols. Furthermore, the *FLASH* content can be streamed (as in *RTMP*) or just downloaded to the user's computer, and then saved to a permanent file, or played by

Table 12: Classification by the *Content Classification Module* of VBS Dataset

Content	Flows	Correct [%]	Wrong [%]	Unknown [%]
AUDIO	1765	80.11	6.01	13.88
MULTIMEDIA	497	86.12	0.20	13.68
VIDEO	23511	87.27	0.60	12.13
WWW	2355298	86.06	0.00	13.94
UNKNOWN	20952650	98.94	1.06	0.00
All flows	23333721	97.62	0.96	1.42
Average from classes		87.70	1.58	10.73

the browser (as in the case of YouTube videos, which use *HTTP*). Furthermore, we do not have knowledge about the service provider names.

OpenDPI. This DPI tool was an open-source fork of PACE, with removed support for encrypted protocols and optimization functions. Its accuracy on the *application* level in our case was 54.19 %. Therefore, the range of the values provided by OpenDPI is almost the same as returned by PACE.

nDPI. This DPI tool is an open-source fork of OpenDPI, which was extended to support some of the protocols, which were removed in OpenDPI. Its accuracy on the *application* level was in our case 55.76 %. Furthermore, it is able to provide the classification on the *service provider* level, as *facebook*, *google*, or *twitter*. However, the final output of the classification is mixed on different levels. For some flows we only obtain the application name (as *dns* or *bittorrent*), for some we only obtain the content (as *flash*), and for some we only obtain the service provider name (as *facebook*). Based on the application name we cannot estimate what is the service provider or the content, and vice versa.

Libprotoident. This tool presents a method called Lightweight Packet Inspection (LPI) [23], as it inspects only first four Bytes of payload in each direction. Therefore, it can be used in many places, where the collected payload must be truncated due to privacy reasons or legal issues. Its accuracy on the *application* level in our case was 94.04 %, which means that the tool achieved the highest accuracy among all of the tested DPI tools. The output from the classifier seems to be also structured in an interesting way, since for many application protocols it gives also information about the transport-layer protocol (as *DNS_TCP*, *BitTorrent_UDP*, or *Unknown_UDP*), which is also unique among all the tested DPI tools. However, many flows obtain the classification only on the *content* level (as *Flash_Player*), or the *service provider* level (as *YahooError*).

NBAR. Network-Based Application Recognition (NBAR) was added to many Cisco devices as a feature, which was supposed to work together with other functions of

these devices, as traffic filtering or shaping. This tool combines different techniques, as port-based classification and DPI. Its accuracy on the *application* level was in our case 25.18 %. This tool provides a very consistent output, as all the results are given on the *application* level. Despite that, the classification accuracy of other tools tested by us was higher.

L7-Filter. L7-filter [24] is an open-source combination of the DPI engine and rules, which can be downloaded separately and applied to the traffic in various configurations, depending on their status and order. Since the last version of the rules is from 2009, the accuracy on the application level was in our case 16.19–30.21 % depending on the configuration. As we observed, the results were returned always on the application level.

UPC Classifier. Another interesting classification approach is shown by authors of a tool developed at UPC BarcelonaTech [25]. The authors performed extensive work regarding traffic classification by various MLAs (including C5.0). They succeeded in comparing various classification techniques in distinguishing of 14 different application classes, and they achieved accuracy of 88-97 %. The final classification tool is based on C5.0 and it uses an automatic retraining system. It is shown to provide in the particular network results, which are comparable to the results achieved by PACE. The ground truth for the training data is established based on DPI techniques: at first, the class of a flow is tried to be obtained by PACE. If PACE cannot provide the result, L7-filter and nDPI are used. Only the cases for which the ground truth is obtained are provided as the training data, so the classifier does not provide any *UNKNOWN* results – all the cases are assigned to the most probable class. Besides that, the classifier also uses some time-based parameters, as inter-arrival times of packets or duration of the flow, so the results can depend on the conditions in the network. The format of the results is a mix of the formats of results provided by PACE, L7-filter, and nDPI.

In our study, we tried to cover the biggest possible extent of applications and their behaviors. It has, however, many limitations:

- Our classifier is able to recognize only 16 applications and 14 different behaviors in total. Deep Packet Inspection tools as well as automatically trained classifiers are able to recognize thousands of different applications. The construction of the multilevel classifier requires us to manually create rules for each application (or group of applications) and for each of its behaviors. Therefore, we could include in our study only these applications, from which we collected sufficient amount of traffic. Nevertheless, our classification covers nearly 90 % of the total data volume in the dataset with full payloads and 85 % of the data volume in the second dataset.
- Our datasets contain limited number of applications. The first dataset, which includes full payloads, was created by us using 3 virtual machines and running the selected applications. The second dataset contains the data obtained from

54 users from Denmark and Poland over 1.5 years. However, the users can be considered as not being enough representative – half of them are students using computers in a school, and the majority of the rest are highly qualified network users.

- The amount and the character of the flows for which we established the ground truth also has an impact on the overall accuracy. As shown, we were not able to establish the ground truth for around 60 % of flows in the first dataset and for 26 % of flows in the second dataset. These flows in majority contain less than 10 packets, usually 2–4, so the socket name is not noticed by VBS. Therefore, our results should be considered as results of classification of longer flows.
- For the second dataset (without packet payloads), we were not able to evaluate the classifier on the *Ethernet*, *IP protocol*, and *service provider* levels. Although the classifier should be always accurate on the *Ethernet* and *IP protocol* levels, the accuracy of the classification on the *service provider* level is a big question mark.
- Considering a big variety of different types of contents and service providers, we did not study how much training is needed for the classifier to provide high accuracy in a real environment. We neither know, what is the accuracy of the classifier trained in one environment and working in another one.
- Constructing the rules used to establish the ground truth for the selected applications and their behaviors is a time-consuming task. The time spent on that can be compared with the time needed to construct DPI rules.

7 Related Work

7.1 Classification Approaches Using MLAs

Usage of Machine Learning Algorithms (MLAs) in traffic classification is broadly covered by existing scientific literature. A comprehensive survey of various Machine Learning-based classification approaches is done in [26]. In [4], the authors succeeded in distinguishing 12 different applications by C4.5 based on first 5 packets in the flow with accuracy of around 97 %. J48 (a Java implementation of C4.5 with further improvements) was used in [5] to classify 5 different applications. The authors showed that it is possible to skip from 10 to 1000 packets from the beginning of a flow without significant decrease of the accuracy, which was fluctuating around 96 %. The attribute set used in this experiment included time-based attributes like flow duration and inter-arrival time. In [27], J48 was used to detect FTP and BitTorrent traffic based on size-dependent attributes, and the authors demonstrated that encryption mechanisms do not influence accuracy of the classification, which reached around 98 %.

In [3], the authors achieved 96-99 % of accuracy, while distinguishing traffic belonging to 5 different classes by C4.5. The classification attributes included those that require the possession of whole flows (like flow duration) as well as time-based attributes (as inter-arrival time distribution). It is worth mentioning that the training data were created by pre-classification based on ports. Flow duration as classification attribute was also used in [10], where the authors identified 6 applications with accuracy around 88 % using C4.5. The method described in [28] tries to recognize different kinds of HTTP traffic, including video progressive download, based on flows groups. This interesting Machine Learning approach uses Weka software. Another approach for traffic classification is described in [29] and is based on signatures contained by packets belonging to flows of particular types. Signature matching also requires processing of whole flows (at least until the signature is matched).

7.2 Approaches for Obtaining Training Data

The accuracy and results of traffic classification by MLAs are heavily dependent on the quality of the data based on which the classifier was trained. Inaccurate training data can result in many problems, as low classification accuracy, or high accuracy, but inappropriate classification of test cases (if training cases were pre-classified incorrectly). In all the papers referenced above, the training data were obtained in one of the following ways: collecting data from one application a time, port-based classification, DPI, statistical classification, or using public data traces. Drawbacks of most of the methods were already described. Collecting data from one application at a time, for example using Wireshark, is time-consuming, not scalable, and it requires a good separation of background traffic, such as DNS requests, or system updates.

Obtaining the ground truth can be based on already existing datasets. An example are Cooperative Association for Internet Data Analysis (CAIDA) data traces, which were collected in a passive or an active way [30]. Another example is the Internet Measurement Data Catalog [31], also operated by CAIDA, which provides the references to different sources of data traces, which are available for research. The data are not stored by CAIDA itself, but on external servers [32]. Although the datasets are pre-classified (or they claim to contain only the traffic from the particular application / protocol), we do not know how the sets were created and how clean they are, which is a very important factor during testing traffic classifiers. MAWI repository [33] contains various packet traces, including daily 15-minutes traces made at a trans-Pacific line (150 Mbit/s link). The traces contain the first 96 bytes of the payload and they were used in several publications, including the ones about testing of different traffic classifiers [34]. However, their usefulness in testing of classifiers is quite limited since we do not know what they consist of. Another useful data source is the Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD) [35], which stores wireless trace data from many contributing locations. Some interesting comparison studies were made

using datasets from different providers. In [36], the authors compare the data obtained from CAIDA and CERNET [37]. Many significant differences between them were found and they concern the lifetimes, lengths, rates of the flows, and the distribution of the TCP and UDP ports among them. Another interesting project is The Waikato Internet Traffic Storage (WITS) [38], which aims to collect and document all the Internet traces that the WAND Network Research Group from the University of Waikato has in their possession. Some of the traces can be freely downloaded and they contain traffic traces from various areas and of different types (as DSL residential traffic, university campus traffic, etc). Most of the traces do not have payload (it is zeroed or truncated).

7.3 Our Previous Approaches to Traffic Classification

To avoid problems with establishing of the ground truth for the use of the training data, at Aalborg University we constructed a new tool for collecting of network data, Volunteer-Based System (VBS), which relies on data collected from users, who agree to install the VBS client on their computers. The current architecture and implementation of VBS was presented in [15]. The system was introduced in Section 3.1, as it was also used in our current approach for establishing the ground truth. We conducted several experiments and invented a few methods for traffic classification and Quality of Service (QoS) assessment using the data collected by VBS [39–42]. Other examples of usage of the VBS tool were obtaining statistics on the flow, application, and content levels [43, 44], obtaining the ground truth for comparison of different DPI tools [20, 21], or a direct comparison of the DPI tools [45].

As the classification tool in all of our experiments, we chose C5.0, as this improved version of C4.5 is characterized by better performance and accuracy [46]. Our first approach involved recognizing traffic belonging to 7 different traffic groups: Skype, FTP, BitTorrent, web browsing, web radio, interactive gaming, and SSH. We achieved accuracy of over 99 % dependent on the classification options, by using only 35 packets from a random point in the flow [39]. However, in this experiment we did not sub-classify the traffic inside the *web browsing* class, which can be characterized by different behavior and which can carry various contents: regular website browsing, audio streaming, video streaming, file downloads, etc. Therefore, we updated VBS and included the possibility of inspecting the *content-type* header in HTTP packets. We found out that one transport-layer flow can contain multiple application-layer HTTP streams, which transmit various types of content: HTML files, web images, audio files, etc. The first packet (and only first packet) of each HTTP stream contains the *content-type* header, which indicates the type of the content. In [42], we demonstrated how to use this information in classification of HTTP traffic by C5.0. This initial approach had overall classification accuracy of around 85 %, but we observed poor accuracy between some traffic classes, e.g. 30 % of error between *video* and *file transfer*. Later, we found out that this was caused by including in the *video* class not only the streamed content, but also video

content, which was downloaded to the users' computers by HTTP (for example, from YouTube). Other concerns include the problem with precisely defining traffic behavior, as web browsing. The disability to deal with the unknown traffic (other than matched by our classes) was the last important issue, and the show stopper at the same time – without resolving this issue, the solution could not be implemented in the real network.

7.4 Traffic Identification by Clustering

Usage of MLAs in recognition of computer network traffic by clustering is less popular than classification. The most commonly used clustering algorithms are K-Means, Density-Based Spatial Clustering of Applications with Noise (DBSCAN), Expectation Maximization (EM), and other based on them [47–54]. As the environment for performing experiments, the scientist most often choose Waikato Environment for Knowledge Analysis (WEKA), which provides support for K-Means, COBWEB, DBSCAN, EM, Farthest First, Ordering Points to Identify the Clustering Structure (OPTICS), and many other clustering techniques. All these algorithms in the context of WEKA were described and compared in [55]. The authors have noticed that DBSCAN and EM algorithms do not require that the researchers know the number of clusters before the experiments starts (contrary to K-Means), and therefore, they are the most useful clustering tools to resolve real life problems. K-Means and DBSCAN were considered for TCP traffic classification of 8 different applications in [47]. Pre-classified public data traces (1000 and 2000 samples per category) were used as the input. The overall accuracy of clustering was assessed depending on the chosen data traces to be 79 % and 84 % for K-Means, while for DBSCAN it was 76 % and 72 %.

Another approach, which is able to detect both seen and unseen yet applications by Particle Swarm Optimization (PSO) algorithm was made in [48]. The authors found out that 5000 samples in the training data are sufficient to obtain the clustering accuracy of 7 different applications of around 95 %. The other papers, which describe the clustering techniques in traffic classification, also deal with quite low number of cases provided to the algorithm, for example, 5000 samples in total in [49] and 500 samples per application in [54]. The performance of the available clustering algorithms was not assessed while trying to construct the clusters from big amounts of data, which is necessary while trying to create an effective classifier for the core network traffic.

8 Conclusions

This paper introduces a novel hybrid method for traffic classification and accounting, which was created to overcome the drawbacks of already existing methods and tools. The classification is performed on six levels: *Ethernet*, *IP protocol*, *application*, *behavior*, *content*, and *service provider*. The system created based on the method was tested on two datasets and it was shown that it provides accurate results on all the levels. The

classification error did not exceed 1.5 % on any level during the evaluation on both datasets. Since the method does not require to inspect the application-layer packet payloads, the classification is fast and lightweight. So, it can be performed even in high-speed networks. Moreover, the process does not interfere with users' privacy issues. Due to the consistent classification of each flow on all the levels, the accounted traffic can be used to generate many useful reports. The reports can be used by ISPs to improve their services, lower costs, and attract new customers. Our open-source prototype of the system in Java can be downloaded from Sourceforge [12]. The future work will concentrate on introducing support for new applications and their behaviors.

9 Acknowledgments

This research work was conducted in the context of a PhD project in the Section for Networking and Security in the Department of Electronic Systems at Aalborg University. The project is co-financed by the European Regional Development Fund (ERDF)[2], and Bredbånd Nord A/S[3], a regional fiber networks provider. We are grateful, for the possibility to install and test our Volunteer-Based System, to *Gimnazjum nr 3 z Oddziałami Integracyjnymi i Dwujęzycznymi imienia Karola Wojtyły w Mysłowicach*[4], a high school in Poland, and to Bredbånd Nord A/S. Finally, we are also very grateful to Josep Solé-Pareta, a professor from Universitat Politècnica de Catalunya (UPC) in Barcelona – the 3-months PhD stay at UPC in an excellent research environment significantly contributed to the knowledge used in creating this publication.

References

[1] Arthur Callado, Carlos Kamienski, Géza Szabó, B Gero, Judith Kelner, Stênio Fernandes, and Djamel Sadok. A Survey on Internet Traffic Identification. *Communications Surveys & Tutorials, IEEE*, 11(3):37–52, July 2009. DOI: 10.1109/SURV.2009.090304.

[2] Riyad Alshammari and A. Nur Zincir-Heywood. Machine Learning based encrypted traffic classification: identifying SSH and Skype. In *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA 2009)*, pages 1–8. IEEE, Ottawa, Ontario, Canada, July 2009. DOI: 10.1109/CISDA.2009.5356534.

[3] Sven Ubik and Petr Žejdl. Evaluating application-layer classification using a Machine Learning technique over different high speed networks. In *Proceedings*

[2] See http://ec.europa.eu/regional_policy/thefunds/regional/index_en.cfm
[3] See http://www.bredbaandnord.dk/
[4] See http://www.nr3.edu.pl/

of the Fifth International Conference on Systems and Networks Communications (ICSNC), pages 387–391. IEEE, Nice, France, August 2010. DOI: 10.1109/ICSNC.2010.66.

[4] Jun Li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. Real-time P2P traffic identification. In *Proceedings of the IEEE Global Telecommunications Conference (IEEE GLOBECOM 2008)*, pages 1–5. IEEE, New Orleans, Louisiana, USA, December 2008. DOI: 10.1109/GLOCOM.2008.ECP.475.

[5] Ying Zhang, Hongbo Wang, and Shiduan Cheng. A Method for Real-Time Peer-to-Peer Traffic Classification Based on C4.5. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 1192–1195. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689126.

[6] Jing Cai, Zhibin Zhang, and Xinbo Song. An analysis of UDP traffic classification. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 116–119. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689203.

[7] Riyad Alshammari and A. Nur Zincir-Heywood. Unveiling Skype encrypted tunnels using GP. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, Barcelona, Spain, July 2010. DOI: 10.1109/CEC.2010.5586288.

[8] L7-filter Supported Protocols, 2012. [Online]. Available: `http://l7-filter.sourceforge.net/protocols`.

[9] Li Jun, Zhang Shunyi, Lu Yanqing, and Zhang Zailong. Internet Traffic Classification Using Machine Learning. In *Proceedings of the Second International Conference on Communications and Networking in China (CHINACOM '07)*, pages 239–243. IEEE, Shanghai, China, August 2007. DOI: 10.1109/CHINACOM.2007.4469372.

[10] Yongli Ma, Zongjue Qian, Guochu Shou, and Yihong Hu. Study of Information Network Traffic Identification Based on C4.5 Algorithm. In *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '08)*, pages 1–5. IEEE, Dalian, China, October 2008. DOI: 10.1109/WiCom.2008.2678.

[11] Wei Li and Andrew W. Moore. A Machine Learning Approach for Efficient Traffic Classification. In *Proceedings of the Fifteenth IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'07)*, pages 310–317. IEEE, Istanbul, Turkey, October 2007. DOI: 10.1109/MASCOTS.2007.2.

[12] Volunteer-Based System for Research on the Internet, 2012. [Online]. Available: http://vbsi.sourceforge.net/.

[13] IEEE Public EtherType list, 2013. [Online]. Available: http://standards.ieee.org/develop/regauth/ethertype/eth.txt.

[14] Protocol Numbers, 2013. [Online]. Available: http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml.

[15] Tomasz Bujlow, Kartheepan Balachandran, Sara Ligaard Nørgaard Hald, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for research on the Internet traffic. *TELFOR Journal*, 4(1):2–7, September 2012. Accessible: http://journal.telfor.rs/Published/Vol4No1/Vol4No1.aspx.

[16] Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for classification of traffic in computer networks. In *Proceedings of the 19th Telecommunications Forum TELFOR 2011*, pages 210–213. IEEE, Belgrade, Serbia, November 2011. DOI: 10.1109/TELFOR.2011.6143528.

[17] Kartheepan Balachandran and Jacob Honoré Broberg. Volunteer-Based Distributed Traffic Data Collection System. Master's thesis, Aalborg University, Department of Electronic Systems, Denmark, June 2010.

[18] De Sensi, Daniele and Danelutto, Marco and Deri, Luca. Dpi over commodity hardware: implementation of a scalable framework using fastflow. Master's thesis, Università di Pisa, Italy, 2012. Accessible: http://etd.adm.unipi.it/t/etd-02042013-101033/.

[19] Tomasz Bujlow and Jens Myrup Pedersen. A Practical Method for Multilevel Classification and Accounting of Traffic in Computer Networks. Technical report, Section for Networking and Security, Department of Electronic Systems, Aalborg University, February 2014. Accessible: http://vbn.aau.dk/da/persons/tomasz-bujlow(c70a43e3-4879-4fd6-89fd-d4679bd168cf)/publications.html.

[20] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification. Technical report, Department of Computer Architecture (DAC), Universitat Politècnica de Catalunya (UPC), June 2013. Accessible: https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2013,en.html.

[21] Valentín Carela-Español, Tomasz Bujlow, and Pere Barlet-Ros. Is our ground-truth for traffic classification reliable? In *Proceedings of the 15th Passive and Active Measurement Conference (PAM 2014), Proceedings Series: Lecture Notes in Computer*

Science 8362, pages 98–108. Springer International Publishing Switzerland, Los Angeles, USA, March 2014. DOI: 10.1007/978-3-319-04918-2_10.

[22] PACE | Network Analysis with Layer-7 Deep Packet Inspection, 2013. [Online]. Available: http://www.ipoque.com/en/products/pace.

[23] Shane Alcock and Richard Nelson. Libprotoident: Traffic Classification Using Lightweight Packet Inspection. Technical report, University of Waikato, August 2012. Accessible: http://www.wand.net.nz/publications/lpireport.

[24] Application Layer Packet Classifier for Linux, 2009. [Online]. Available: http://l7-filter.sourceforge.net/.

[25] Oriol Mula-Valls. A practical retraining mechanism for network traffic classification in operational environments. Master's thesis, Computer Architecture, Networks and Systems, Universitat Politècnica de Catalunya, Spain, June 2011. Accessible: https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2011,en.html.

[26] Thuy T. T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4):56–76, October 2008. DOI: 10.1109/SURV.2008.080406.

[27] Jason But, Philip Branch, and Tung Le. Rapid identification of BitTorrent Traffic. In *Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN)*, pages 536–543. IEEE, Denver, Colorado, USA, October 2010. DOI: 10.1109/LCN.2010.5735770.

[28] Kei Takeshita, Takeshi Kurosawa, Masayuki Tsujino, Motoi Iwashita, Masatsugu Ichino, and Naohisa Komatsu. Evaluation of HTTP video classification method using flow group information. In *Proceedings of the 14th International Telecommunications Network Strategy and Planning Symposium (NETWORKS)*, pages 1–6. IEEE, Warsaw, Poland, September 2010. DOI: 10.1109/NETWKS.2010.5624929.

[29] Samruay Kaoprakhon and Vasaka Visoottiviseth. Classification of audio and video traffic over HTTP protocol. In *Proceedings of the 9th International Symposium on Communications and Information Technology (ISCIT 2009)*, pages 1534–1539. IEEE, Icheon, Korea, September 2009. DOI: 10.1109/ISCIT.2009.5341026.

[30] CAIDA Internet Data – Passive Data Sources, 2012. [Online]. Available: http://www.caida.org/data/passive/.

[31] CAIDA Internet Data – Internet Measurement Data Catalog (IMDC), 2012. [Online]. Available: http://www.caida.org/projects/trends/imdc/.

[32] Colleen Shannon, David Moore, Ken Keys, Marina Fomenkov, Bradley Huffaker, and Kimberly C. Claffy. The internet measurement data catalog. *ACM SIGCOMM Computer Communication Review*, 35(5):97–100, October 2005. DOI: 10.1145/1096536.1096552.

[33] MAWI Working Group Traffic Archive, 2013. [Online]. Available: http://mawi.wide.ad.jp/mawi/.

[34] Kensuke Fukuda. Difficulties of identifying application type in backbone traffic. In *2010 International Conference on Network and Service Management (CNSM)*, pages 358–361. IEEE, Niagara Falls, Ontario, Canada, October 2010. DOI: 10.1109/CNSM.2010.5691234.

[35] CRAWDAD, 2013. [Online]. Available: http://crawdad.cs.dartmouth.edu/.

[36] Xiaoguo Zhang and Wei Ding. Comparative Research on Internet Flows Characteristics. In *2012 Third International Conference on Networking and Distributed Computing (ICNDC)*, pages 114–118. IEEE, Hangzhou, China, October 2012. DOI: 10.1109/ICNDC.2012.35.

[37] CERNET data traces, 2012. [Online]. Available: http://ntds.njnet.edu.cn/data/index.php.

[38] WITS: Waikato Internet Traffic Storage, 2013. [Online]. Available: http://www.wand.net.nz/wits/.

[39] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for classification of network traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of ICNC'12: 2012 International Conference on Computing, Networking and Communications (ICNC): Workshop on Computing, Networking and Communications*, pages 244–248. IEEE, Maui, Hawaii, USA, February 2012. DOI: 10.1109/ICCNC.2012.6167418.

[40] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for Assessing Quality of Service in Broadband Networks. In *Proceedings of the 14th International Conference on Advanced Communication Technology (ICACT)*, pages 826–831. IEEE, Phoenix Park, PyeongChang, Korea, February 2012. Accessible: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6174795.

[41] Tomasz Bujlow, Sara Ligaard Hald, M. Tahir Riaz, and Jens Myrup Pedersen. A Method for Evaluation of Quality of Service in Computer Networks. *ICACT Transactions on the Advanced Communications Technology (TACT)*, 1(1):17–25, July 2012. Accessible: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6488383.

[42] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. Classification of HTTP traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of the Fourth IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Web-based Service Architectures (PEDISWESA 2012)*, pages 882–887. IEEE, Cappadocia, Turkey, July 2012. DOI: 10.1109/ISCC.2012.6249413.

[43] Jens Myrup Pedersen and Tomasz Bujlow. Obtaining Internet Flow Statistics by Volunteer-Based System. In *Fourth International Conference on Image Processing & Communications (IP&C 2012), Image Processing & Communications Challenges 4, AISC 184*, pages 261–268. Springer Berlin Heidelberg, Bydgoszcz, Poland, September 2012. DOI: 10.1007/978-3-642-32384-3_32.

[44] Tomasz Bujlow and Jens Myrup Pedersen. Obtaining Application-based and Content-based Internet Traffic Statistics. In *Proceedings of the 6th International Conference on Signal Processing and Communication Systems (ICSPCS'12)*, pages 1–10. IEEE, Gold Coast, Queensland, Australia, December 2012. DOI: 10.1109/ICSPCS.2012.6507984.

[45] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Extended Independent Comparison of Popular Deep Packet Inspection (DPI) Tools for Traffic Classification. Technical report, Department of Computer Architecture (DAC), Universitat Politècnica de Catalunya (UPC), January 2014. Accessible: https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2014,en.html.

[46] Information on See5/C5.0 – RuleQuest Research Data Mining Tools, 2011. [Online]. Available: http://www.rulequest.com/see5-info.html.

[47] Zhou Xusheng and Zhou Yu. Application Of Clustering Algorithms In IP Traffic Classification. In *Proceedings of the WRI Global Congress on Intelligent Systems (GCIS'09)*, volume 2, pages 399–403. IEEE, Xiamen, China, May 2009. DOI: 10.1109/GCIS.2009.139.

[48] Liu Bin and Tu Hao. An Application Traffic Classification Method Based on Semi-Supervised Clustering. In *Proceedings of the 2nd International Symposium on Information Engineering and Electronic Commerce (IEEC)*, pages 1–4. IEEE, Ternopil, Ukraine, July 2010. DOI: 10.1109/IEEC.2010.5533239.

[49] Xiang Li, Feng Qi, Li Kun Yu, and Xue Song Qiu. High accurate Internet traffic classification based on co-training semi-supervised clustering. In *Proceedings of the International Conference on Advanced Intelligence and Awareness Internet (AIAI 2010)*, pages 193–197. IET, Beijing, China, October 2010. DOI: 10.1049/cp.2010.0751.

[50] Jing Yuan, Zhu Li, and Ruixi Yuan. Information Entropy Based Clustering Method for Unsupervised Internet Traffic Classification. In *Proceedings of the IEEE International Conference on Communications (ICC'08)*, pages 1588–1592. IEEE, Beijing, China, May 2008. DOI: 10.1109/ICC.2008.307.

[51] Thuy T. T. Nguyen and Grenville Armitage. Clustering to Assist Supervised Machine Learning for Real-Time IP Traffic Classification. In *Proceedings of the IEEE International Conference on Communications (ICC'08)*, pages 5857–5862. IEEE, Beijing, China, May 2008. DOI: 10.1109/ICC.2008.1095.

[52] Caihong Yang and Benxiong Huang. Traffic classification using an improved clustering algorithm. In *Proceedings of the International Conference on Communications, Circuits and Systems (ICCCAS 2008)*, pages 515–518. IEEE, Xiamen, Fujian, China, May 2008. DOI: 10.1109/ICCCAS.2008.4657826.

[53] Liu Yingqiu, Li Wei, and Li Yunchun. Network Traffic Classification Using K-means Clustering. In *Proceedings of the Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*, pages 360–365. IEEE, Iowa City, Iowa, USA, August 2007. DOI: 10.1109/IMSCCS.2007.52.

[54] Pedro Casas, Johan Mazel, and Philippe Owezarski. On the use of Sub-Space Clustering & Evidence Accumulation for traffic analysis & classification. In *Proceedings of the 7th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1016–1021. IEEE, Istanbul, Turkey, July 2011. DOI: 10.1109/IWCMC.2011.5982680.

[55] Narendra Sharma, Aman Bajpai, and Ratnesh Litoriya. Comparison the various clustering algorithms of weka tools. *International Journal of Emerging Technology and Advanced Engineering*, 2(5):73–80, May 2012.

Tomasz Bujlow is working as a PhD Student in the Section for Networking and Security (NetSec) in the Department of Electronic Systems at Aalborg University in Denmark. He received the degree of Master of Science in Computer Engineering from Silesian University of Technology in Poland in 2008, specializing in databases, computer networks, and computer systems. He also obtained the degree of Bachelor of Computer Engineering from University of Southern Denmark in 2009, specializing in software engineering and system integration. His research interests include methods for measurement of Quality of Service and traffic classification in computer networks. He is also a Cisco Certified Network Professional (CCNP) since 2010.

Jens Myrup Pedersen is working as an Associate Professor and the head of the Section for Networking and Security (NetSec) in the Department of Electronic Systems at Aalborg University. His current research interests include network planning, traffic monitoring, and network security. He obtained the degree of Master of Science in Mathematics and Computer Science from Aalborg University in 2002, and PhD degree in Electrical Engineering also from Aalborg University in 2005. He is an author/co-author of more than 80 publications in international conferences and journals, and has participated in Danish, Nordic and European funded research projects. He is also a board member of a number of companies within technology and innovation.

Paper IX

Independent Comparison of Popular DPI Tools for Traffic Classification

Tomasz Bujlow[a], Valentín Carela-Español[b], and Pere Barlet-Ros[b]

[a] *Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark*
tbu@es.aau.dk

[b] *Broadband Communications Research Group, Department of Computer Architecture, Universitat Politècnica de Catalunya, ES-08034, Barcelona, Spain*
{vcarela, pbarlet}@ac.upc.edu

This paper[1] is submitted for review to *Computer Networks* – a journal from Elsevier.

[1] This research was funded by the Spanish Ministry of Science and Innovation under contract TEC2011-27474 (NOMADS project) and by the Comissionat per a Universitats i Recerca del DIUE de la Generalitat de Catalunya (ref. 2009SGR-1140). This work was also co-financed (grant no. 8–10100) by Aalborg University, the European Regional Development Fund (ERDF), and Bredbånd Nord A/S.

Abstract

Deep Packet Inspection (DPI) is the state-of-the-art technology for traffic classification. According to the conventional wisdom, DPI is the most accurate classification technique. Consequently, most popular products, either commercial or open-source, rely on some sort of DPI for traffic classification. However, the actual performance of DPI is still unclear to the research community, since the lack of public datasets prevent the comparison and reproducibility of their results. This paper presents a comprehensive comparison of 6 well-known DPI tools, which are commonly used in the traffic classification literature. Our study includes 2 commercial products (*PACE* and *NBAR*) and 4 open-source tools (*OpenDPI*, *L7-filter*, *nDPI*, and *Libprotoident*). We studied their performance in various scenarios (including packet and flow truncation) and at different classification levels (application protocol, application and web service). We carefully built a labeled dataset with more than 750 K flows, which contains traffic from popular applications. We used the Volunteer-Based System (VBS), developed at Aalborg University, to guarantee the correct labeling of the dataset. We released this dataset, including full packet payloads, to the research community. We believe this dataset could become a common benchmark for the comparison and validation of network traffic classifiers. Our results present *PACE*, a commercial tool, as the most accurate solution. Surprisingly, we find that some open-source tools, such as *nDPI* and *Libprotoident*, also achieve very high accuracy.

Keywords

Deep Packet Inspection, PACE, nDPI, Libprotoident, NBAR, L7-filter

1 Introduction

The payload of the packets contains an enormous amount of information. Deep Packet Inspection (DPI) technologies exploit this source of information by searching for specific patterns (i.e., signatures) in the content of the packets. It is thus not surprising that DPI became a fundamental technology for many applications, including network management, intrusion detection, and network forensics, to mention just a few examples.

One of the most prolific applications of the DPI technology is in the area of network traffic classification. Generally, DPI-based techniques extract, in an offline phase, a set of characteristic signatures of network applications and services (e.g., Skype, BitTorrent, Facebook, YouTube) from the payload of the packets they generate. These signatures are later employed, usually in an online phase, to classify the traffic flowing in a network. However, this process is very complex. On the one hand, the extraction of signatures is a difficult task that should be periodically performed in order to adapt

them to the continuous evolution of the applications (e.g., new applications, new versions, new obfuscation techniques). On the other hand, pattern matching algorithms for online classification are computationally complex and usually require expensive hardware. Even so, DPI is commonly considered as the most accurate technique for traffic classification, and most commercial solutions rely on it [1–4].

Although the state-of-the-art proposals in the traffic classification field show very accurate results, the network classification problem is far from being totally solved. As already pointed out in [5], an important aspect that remains unsolved is the validation and comparison of the existing techniques for traffic classification. One of the main problems is that this validation directly depends on the techniques used to set the ground-truth. This task is usually carried out in the literature by DPI techniques given their presumably high accuracy. However, the actual accuracy of DPI-based techniques is still not clear. Previous works that tried to compare the performance of different DPI techniques showed that the ground-truth used to validate the proposals was obtained through port-based techniques, other DPI-based techniques, or methodologies of unknown reliability [6–10]. Thus, making the results of the comparison arguable. In addition, most commercial tools are black boxes that claim high accuracy based on their own studies, which cannot be validated because they were performed using private datasets.

The first step in the proper validation of traffic classification techniques would be the use of publicly available datasets. In this way, its comparison with other proposals would be easier. An example are the Cooperative Association for Internet Data Analysis (CAIDA) [11] or the Internet Measurement Data Catalog [12]. Although the datasets are pre-classified, the actual reliability of this labeling is unknown, which is a very important factor during testing traffic classifiers. Also, most of them have no payload or just the first bytes of each packet. The MAWI repository [13] contains various packet traces, including daily 15-minutes traces made at an trans-Pacific line (150 Mbit/s link). The bandwidth of this link has changed through the years. The traces contain the first 96 bytes of the payload and the traffic is usually asymmetric. Another useful data source is the Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD) [14], which stores wireless trace data from many contributing locations. Another interesting project is The Waikato Internet Traffic Storage (WITS) [15], which aims to collect and document all the Internet traces that the WAND Network Research Group from the University of Waikato has in their possession. Some of the traces can be freely downloaded and they contain traffic traces from various areas and of different types (as DSL residential traffic, university campus traffic, etc). Most of the traces do not have payload (i.e., it is zeroed) or truncated.

All these datasets although useful for many network evaluations are of limited interest for DPI validation given that no correct labeling can be performed to them.

In [16], it was introduced a method for validation of classification algorithms, which is independent of other classification methods, deterministic, and allows to automatize

testing of large data sets. The authors developed a Windows XP driver based on the Network Driver Interface Specification (NDIS) library. Another approach to obtain the ground-truth was taken in [17]. The authors created a tool, which collects the data from the network and labels the flows with the real application names (e.g., *Thunderbird*) and application protocol names (e.g., *SMTP*). This tool is somehow similar to VBS, the tool used in this work for the ground-truth generation and further described in Section 3.1.1. Yet another way of establishing the ground-truth was shown in [18], which describes a system developed to accelerate the manual verification process. The authors proposed Ground Truth Verification System (GTVS) based on the DPI signatures derived from the databases available in the Internet, including L7-filter. GTVS, however, does not collect the application names from the operating systems, so the established truth cannot be completely verified.

In a previous conference paper [19], we tried to face the problem of ground-truth reliability. We described various methods of obtaining the ground-truth for testing various traffic classification tools. As part of the evaluation, we tested several DPI-based traffic classifiers and assessed if they can be used for obtaining reliable ground-truth. This paper is not only an extension but a more broad and comprehensive validation of DPI-based techniques. The focus of this paper is different, as we directly compare the selected DPI tools regarding their per-class accuracy. Reference datasets for the use in this paper as well as the previous one are generated by the same methodology further explained in Section 3. However, both datasets are completely different. The dataset used in this paper is significantly larger than the one used in the conference paper. Furthermore, the new dataset also contains labeled non-HTTP flows belonging to various web services, which is a unique feature. The methodology of testing the classifiers is also different. In our previous paper, we tested the accuracy of the classifiers on a single level. In the current paper, we evaluate different levels (i.e., application, web service, etc), so the new evaluation method is more detailed and complete.

This paper compares and validates six well-known DPI-based tools used for network traffic classification. In order to allow the validation of our work, we publish the reliable labeled dataset used to perform our study. Two main aspects have been carefully addressed when building this dataset: the reliability of the labeling and the representativeness of the data. We used the VBS tool [20] to guarantee the correctness of the labeling process. This tool, described in Section 3, is able to label the flows with the name of the process that creates them. This allowed us to carefully create a reliable ground-truth that can be used as a reference benchmark for the research community to compare other proposals. The selection of applications for our dataset was made based on well-known indexes of the most commonly used Internet applications and web services. In order to allow the publication of the dataset and avoid any privacy issues, we created the traffic by running a large set of applications and meticulously simulating common behaviors of the users.

The main contributions of this paper can be summarized as follows:

- We publish a reliable labeled dataset with full packet payloads. The dataset, further described in Section 4, contains traffic from a diverse set of commonly used applications. Although artificially created, we carefully simulated human behaviors in order to produce a dataset that is as much realistic as possible.
- We compare six well-known DPI-based tools widely used for network traffic classification. Using the previous dataset, we evaluate the precision of PACE, OpenDPI, nDPI, L7-filter, Libprotoident, and NBAR, and compare their results at various classification levels and in different scenarios. We are aware that OpenDPI and L7-filter are abandoned projects, which development stopped several years ago. However, we decided to include them into our evaluation as a reference and for completeness, as many existing scientific papers base their results on these two classifiers.

In this paper, we focus on a single performance parameter: the classification accuracy. We acknowledge that other performance parameters are also important, such as speed, scalability, complexity, robustness, or price of the solutions. However, given that the research community is mainly using DPI-based technique for offline ground-truth generation we think those metrics are less decisive. Furthermore, we believe that our dataset will be useful to evaluate most of these parameters as well.

This work presents an independent and impartial comparison of the most popular DPI-based tools used in the traffic classification literature. As a consequence, our study also provides valuable insights about the reliability of the tools commonly used by researchers to generate their ground truth. The results presented can also help researchers and network managers to better decide which DPI solution is more suitable for their needs and scenarios. Indirectly, we also provide information about the reliability of those non-DPI classification techniques proposed in the literature that used one of the DPI techniques compared in this paper to set their ground truth.

The remainder of this paper is organized as follows. Section 2 describes the DPI-based tools and their configurations used for the evaluation. Section 3 presents the methodology used to obtain the reliable dataset that is described in Section 4. Section 5 presents the results of the performance evaluation of the different DPI-based techniques. Section 6 discusses the results, compares them with the literature, and comments the limitations of our evaluation. Section 7 reviews the related work. Finally, Section 8 concludes and summarizes the outcomes and contributions of the paper.

2 Classification Tools

On the market, there are many available DPI-based traffic classification solutions. For our experiment, we selected PACE, OpenDPI, nDPI, Libprotoident, NBAR, and L7-filter, which will be broadly introduced in this section. Table 1 summarizes these DPI-based tools along their characteristics.

Table 1: DPI Tools Included in Our Comparison

Name	Version	Released	Identified Applications
PACE	1.47.2	November 2013	1000
OpenDPI	1.3.0	June 2011	100
nDPI	rev. 7543	April 2014	170
L7-filter	2009.05.28	May 2009	110
Libprotoident	2.0.7	November 2013	250
NBAR	15.2(4)M2	November 2012	85

PACE. It is a proprietary classification library developed by *ipoque* entirely in C, which supports classical DPI (pattern matching), behavioral, heuristic, and statistical analysis. According to its website, PACE is able to detect encrypted protocols as well as protocols which use obfuscation. Overall, more than 1000 applications and 200 network protocols are supported. It is also possible to include user-defined rules for detection of applications and protocols. To the best of our knowledge, PACE is the only commercial tool used in the literature to build the ground truth [8].

OpenDPI. It was an open-source classifier derived from early versions of PACE by removing support for encrypted protocols, as well as all performance optimizations. The project is now considered as closed. In [6, 7], the authors mention that OpenDPI is not a classic DPI tool, as it uses other techniques apart from pattern matching (i.e., behavioral and statistical analysis). Thanks to that, it should not provide false classification results, but some traffic can remain unclassified [6]. Another interesting feature is flow association, which relies on inspecting the payload of a known flow to discover a new flow, as inspecting a control FTP session to obtain the five tuple of the newly initiated data session [10].

nDPI. It is an OpenDPI fork, which is optimized and extended with new protocols [21]. It re-introduced support for many encrypted ones due to analysis of session certificates. Overall, nDPI for now supports more than 100 protocols [22]. The current architecture is scalable, but it does not provide the best performance and results: each of the protocols has its own signature scanner, through which the packets are examined. Every packet is examined by each scanner, regardless, if a match was found. If there are multiple matches per flow, the returned value is the most detailed one [10]. Additionally, there is no TCP or IP payload re-assembly, so there is no possibility to detect a signature split into multiple TCP segments / IP packets [22].

Libprotoident. This C library [8] introduces Lightweight Packet Inspection (LPI), which examines only the first four bytes of payload in each direction. That allows to minimize privacy concerns, while decreasing the disk space needed to store the packet

traces necessary for the classification. Libprotoident supports over 200 different protocols and the classification is based on a combined approach using payload pattern matching, payload size, port numbers, and IP matching.

Cisco Network Based Application Recognition (NBAR). It was developed to add the ability to classify the network traffic by using the existing infrastructure [23]. It is able to classify applications, which use dynamic TCP and UDP port numbers. NBAR works with Quality of Service (QoS) features, thanks to what the devices (e.g., routers) can dynamically assign a certain amount of bandwidth to a particular application, drop packets, or mark them in a selected way. The authors claim that NBAR supports a wide range of stateful protocols, which are difficult to classify.

L7-filter. It was created in 2003 as a classifier for Linux Netfilter, which can recognize the traffic on the application layer [24]. The classification is based on three techniques. At first, simple numerical identification based on the standard iptables modules, which can handle port numbers, IP protocol numbers, number of transferred bytes, etc. At second, payload pattern matching based on regular expressions. At third, the applications can be recognized based on functions. L7-filter is developed as a set of rules and a classification engine, which can be used independently of each other. The most recent version of L7-filter classification engine is from January, 2011, and the classification rules from 2009.

3 Methodology

Our experiment involved numerous steps, which will be defined and described in this section. We had two main goals – building the dataset and testing the classifiers. Each of them required an individual methodology.

3.1 Building of the Dataset

3.1.1 Testbed

Our testbed consisted of 7 machines, which were used for running the selected applications and generating the traffic data, and of a server. We equipped the data generating machines with Windows 7 (3 machines), Ubuntu (3 machines), and Windows XP (1 machine). The additional Ubuntu server machine was equipped with a MySQL database for data storage.

To collect and accurately label the flows, we adapted the Volunteer-Based System (VBS) developed at Aalborg University [25]. The goal of the VBS project is to collect flow-level information from the Internet traffic (e.g., start time of the flow, number of packets contained by the flow, local and remote IP addresses, local and remote ports,

transport layer protocol) together with detailed information about each packet (e.g., direction, size, TCP flags, and relative timestamp to the previous packet in the flow). For each flow, the system collects the process name associated with it, which is obtained from the system sockets. Additionally, the system collects some information about the types of the transferred HTTP contents (e.g., *text/html*, *video/x-flv*). The captured information is transmitted to the VBS server, which stores the data in a MySQL database.

On every data generating machine, we installed a modified version of VBS. The source code of the original system as well as the modified version is publicly available in [26] under *GNU General Public License v3.0*. The modified version of the VBS client captures, apart from the data described above, full Ethernet frames for each packet, and extracts the HTTP *URL* and *Referrer* fields – all the information is transmitted to the server, and stored in the MySQL database. We added a module called *pcapBuilder*, which is responsible for dumping the packets from the database to PCAP files. At the same time, INFO files are generated to provide detailed information about each flow, which allows us to assign each packet from the PCAP file to an individual flow.

3.1.2 Selection of the Data

The process of building a representative dataset, which characterizes a typical user behavior, is a challenging task, crucial from the point of testing and comparing different traffic classifiers. Therefore, to ensure the proper diversity and amount of the included data, we decided to combine the data on a multidimensional level. Based on w3schools statistics [27], we found that most PC users use Windows 7 (56.7 % of users), Windows XP (12.4 %), Windows 8 (9.9 %), and Linux (4.9 %) - state for October 2013. Apple computers contribute for 9.6 % of the overall traffic, and mobile devices for 3.3 %. Because of the lack of the equipment and/or software for Apple computers, Windows 8, and mobile devices, we decided to include in our study Windows 7 (W7), Windows XP (XP), and Linux (LX), which cover now 74.0 % of the used operating systems.

The application protocols, applications, and web services selected for this study are shown below. To group them, we adopted the same classification categories used in the reports from Palo Alto [28].

1. File-sharing applications. According to [28], they account for 6 % of the total bandwidth. Inside that group, BitTorrent accounts for 53 %, FTP for 21 %, Dropbox for 5 %, Xunlei for 4 %, and eMule for 3 %. Based on the statistics found in [28], as well as those in the CNET [29] and the OPSWAT P2P clients popularity list, the CNET FTP clients popularity list [30], and the Direct Download popularity list [31], we selected the following applications:

    - BitTorrent: uTorrent (Windows), kTorrent (Linux).
    - eDonkey: eMule (Windows), aMule (Linux). The studied configurations were: outgoing-non-obfuscated-incoming-all, all-obfuscated.

- FTP: FileZilla (Windows, Linux) in active mode (PORT) and passive mode (PASV).
- Dropbox (Windows, Linux).
- Web-based direct downloads: 4Shared (+ Windows app), MediaFire, Putlocker.
- Webdav (Windows).

2. Photo-video group. According to the reports from Palo Alto [28], they account for 16 % of the total bandwidth, where YouTube accounts for 6 % of total, Netflix for 2 % of total, other HTTP video for 2 % of total, RTMP for 2 % of total, and others for 4 % of traffic in total. To select the applications in this category we also used the Ebizmba ranking of video websites [32].

   - YouTube: most watched videos from all the times according to the global ranking [33].
   - RTMP: around 30 random short live video streams (1–10 minutes) were watched from Justin.tv.
   - Vimeo – a web-based photo sharing solution.
   - PPStream (Windows) – P2P streaming video software.
   - Other HTTP video.

3. Web browsing traffic. Based on w3schools statistics [34], the most popular web browsers are: Chrome (48.4 % of users), Firefox (30.2 %), and Internet Explorer (14.3 %). These browsers were used to generate the web traffic. According to the reports from Palo Alto [28], they account for 20 % of the total bandwidth. The selection of the websites was based on Alexa statistics [35], Ebizmba web statistics [36], Quantcast statistics [37], and Ebizmba search engines popularity [38]. In order to make the dataset as representative as possible, we simulated different human behaviors when using these websites. For instance, on Facebook, we log in, interact with friends (e.g., chat, send messages, write in their walls), upload pictures, create events or play games. Similar behaviors were simulated for other popular web services, such as Twitter, Google+, eBay, etc. The detailed description of actions performed with the services is listed in our technical report [39].

4. Encrypted tunnel traffic. According to the reports from Palo Alto [28], they account for 9 % of the total bandwidth, where 6 % of total is SSL and 2 % of total is SSH.

   - SSL (Windows, Linux): collected while using various applications and web services.

- SSH (Linux).
- TOR (Windows). First, we used TOR to browse various websites and download big files. Then, we configured TOR to act as an internal relay, so we participated in creating the invisible path for other users.
- Freenet (Windows): for browsing the intranet and also as a relay for other peers.
- SOCKSv5 (Windows). We created a SOCKSv5 server on Linux, and used it for Firefox and uTorrent.

5. Storage-backup traffic. According to Palo Alto [28], they account for 16 % of the total bandwidth, where at least half of the bandwidth is consumed by MS-SMB, and the rest by many different applications. Therefore, the only tested application was MS-SMB (Windows, Linux).

6. E-mail and communication traffic. According to the reports from Palo Alto [28], e-mail traffic accounts for 3 % of the total bandwidth. E-mail market share from October 2013 [40] shows that only one desktop mail client, Microsoft Outlook (17 %), is in the top 10 of used mail clients. The rest is split between web-based clients (as GMail) and mobile clients (Mac, Android). The tested applications / web-based mail services include: Gmail, Hotmail, Windows Live Mail (Windows), and Mozilla Thunderbird (Windows). The desktop e-mail applications (Windows Live Mail and Mozilla Thunderbird) were tested to use various protocols: SMTP-PLAIN (port 587), SMTP-TLS (port 465), POP3-PLAIN (port 110), POP3-TLS (port 995), IMAP-STARTTLS (port 143), and IMAP-TLS (port 993). We also tested Skype between Windows and Android OS: video sessions, voice conversations, and file transfers.

7. Management traffic. DNS, ICMP, NETBIOS, NTP, RDP.

8. Games. Based on the most played online games in USA according to DFC Intelligence [41], we selected:

   - League of Legends (Windows) – with all launchers.
   - World of Warcraft (Windows) – including all launchers.
   - Pando Media Booster (Windows) – a process added by League of Legends to seed the game installer to other users, which offloads the servers, because the download is performed in the P2P mode. It generates enormous accounts of traffic and fills the connection.
   - Steam – delivers a range of games straight to a computer's desktop. Includes automatic updates, lists of games and prices, posters, plus access to a large number of games. We included Steam on the list as it is a platform for numerous games.

- America's Army – a popular game from Steam.

9. Others. This category includes:

   - Music applications: Spotify, iTunes (Windows).
   - P2P Internet TVs: PPLive, Sopcast (Windows).

3.2 Testing of the DPI Tools

The process of testing different DPI tools is complex and, therefore, we split it into several parts: labeling of the data, the classification process, and analysis of the classification logs. Some of the steps can be different for some DPIs than for the others – in these cases the differences are explicitly highlighted. We evaluate only one performance parameter: the classification accuracy. We acknowledge that other performance parameters are also important, such as speed, scalability, complexity, robustness, or price of the solutions, however, their evaluation is outside the scope of this paper.

3.2.1 Labeling of the Data

All the flows stored in the database need to be properly marked by attaching to them the labels of the applications, application protocols, web services, types of the content, or Internet domains. One flow can be associated with multiple labels. Flows, which are not labeled, are not taken into consideration while extracting them to PCAP files.

We classify the flows at different levels. We start the labeling process by identifying the web flows and assigning them to the selected web services. Every web service is identified by a set of domains. The domains were chosen based on the number of their occurrences in the collected HTTP flows. The HTTP flows are marked with a web service label only if they contain the traffic from the matching domains. In case the flow contains traffic from domains belonging to multiple services (or to domains, which are not assigned to the selected services), the flow is left as unlabeled. The HTTP flows are also marked with the labels of the type of the transmitted content (e.g., *video/x-flv*), if they transmit audio or video. Those flows that are not HTTP belonging to the web services (e.g., SSL) are labeled using an heuristic method as follows. To be recognized as a non-HTTP web flow, the application name associated with the flow should be the name of a web browser (e.g., *chrome*), a name of a web browser plugin (e.g., *plugin-container*, *flashgcplay*), or the name should be missing. Then, we look at the HTTP flows, which were originated from 2 minutes before to 2 minutes after the non-HTTP flow. If all the corresponding (i.e., originated from the same local machine and reaching the same remote host) HTTP flows have a web service label assigned, and the service label is the same for all of the flows, the non-HTTP flow is classified with the same web service label.

Afterwards, we identify the application protocols. The application protocol label is applied only to those flows for which we are sure that transmit the specific application protocol. Finally, we identify the applications. We consider applications and protocols individually because, for example, a web-browser may use many different protocols besides HTTP, or a BitTorrent client can connect to websites to download files using HTTP or SSL, etc.

3.2.2 Classification Process

The packets were extracted into PCAP files in 3 different modes: the normal one, with truncated packets (i.e., Ethernet frames were overwritten by 0s after the 70th byte), and with truncated flows (we extracted only 10 first packets for each flow). We designed a tool, called *dpi_benchmark*, which is able to read the PCAP files and provide the packets one-by-one to PACE, OpenDPI, L7-filter, nDPI, and Libprotoident. All the flows are started and terminated based on the information from the INFO files, which contain the timestamps. After the last packet of the flow is sent to the classifier, the tool obtains the label associated with that flow. The labels are written to the log files together with the flow identifier, which makes us later able to relate the classification results to the original flows in the database.

We used the default configurations of all classifiers except for *L7-filter*, which was evaluated in two different configurations. The first version (*L7-filter-all*) had all the patterns activated , but the patterns marked as *overmatching* by their authors have a low priority. For the second version (*L7-filter-com*) we adapted the methodology proposed in [42], which does not activate the patterns declared as *overmatching* and the default pattern priorities were modified.

Classification by *NBAR* required to build a complete working environment. We did not have any Cisco device that could be used for the experiment. Therefore, we used GNS3 – a graphical framework, which uses Dynamips to emulate our Cisco hardware. We emulated 7200 platform, since this is the only platform supported by GNS3 that can run the newest version of Cisco IOS (version 15), which contains Flexible NetFlow. Previous versions of Cisco IOS contain only traditional NetFlow, which does not support NBAR reporting on the per flow basis. We connected the virtual router to a real computer by using a virtual interface. The router was configured to use Flexible NetFlow with NBAR on the created interface.

Every flow recognized by Flexible NetFlow was tagged by the application name obtained from NBAR. On the computer, we used *tcpreplay* to replay the PCAP files to the router with the maximal speed that did not cause packet loss. At the same time, we used *nfacctd*, which is part of PMACCT tools [43], to capture the Flexible NetFlow records sent by the router to the computer.

The data stored in the classification logs was processed and imported back to the database. We matched the log records to the proper flows in the database using the flow identifier contained by each flow record. NBAR relies on Flexible NetFlow, which

treats the flows in a unidirectional way. It means that we needed to assess the type of the bi-directional flow based on 2 unidirectional flows (inbound and outbound). In most of the cases the label from both unidirectional flows were the same. In a few cases there was only an inbound or an outbound flow, since there were no packets going in the opposite direction. In case, both unidirectional flows existed and the label of each of them was different, the bidirectional flow got the label from the unidirectional flow that accounted for more bytes.

3.3 Analysis of the Results

The method for analysis of the results depend on the level of classification on which the flows were labeled:

- *Application protocol* level, such as DNS, HTTP, or POP3: To consider the classification as correct, the label reported by the classifier must be an application protocol (e.g., DNS, HTTP), but not at a different level (e.g., FLASH, YOUTUBE). This is useful to test if the tool can recognize the specific application protocol. If the result is given at a different level, the flow is considered as *unclassified*. However, the same flow will be classified as *correct* during other tests at different levels, when we for example look for a web service called *YouTube*. Those flows with labels belonging to different application protocols, and services and applications that do not belong to this application protocol are considered as *wrong*.

- *Web service* level, such as Yahoo or YouTube: the classification is considered to be *correct* only if the name of the web service is given. If it is given at a different level, such as HTTP or FLASH, the flow is considered as *unclassified*.

- *Application* level (when the application uses its proprietary application-level protocols). For example, uTorrent and Skype applications can use multiple protocols, including their proprietary protocols called respectively *Skype* and *BitTorrent*, and other protocols, such as *HTTP* or *SSL*. For example, *HTTP* and *SSL* can be used to connect to the web server to download the user's data or advertisements. Therefore, in the case of Skype, flows labeled by DPI tools as *Skype*, *HTTP*, *SSL* are all marked as *correct*.

- *Application* level (when the application does not use its proprietary application-level protocols, but directly uses HTTP, SSL, etc.) It concerns for example Spotify. Then, only the flows marked as *Spotify* are considered to be labeled correctly, as no specific application-level protocol exists for this application, so we expect the application name itself to be identified.

Thanks to this multilevel testing approach, we obtained the knowledge of which classifier is able to provide results on each particular level of classification. This knowl-

Table 2: Application Protocols in the Dataset

Protocol	Number of Flows	Number of Megabytes
DNS	18251	7.66
HTTP	43127	7325.44
ICMP	205	2.34
IMAP-STARTTLS	35	36.56
IMAP-TLS	103	410.23
NETBIOS Name Service	10199	11.13
NETBIOS Session Service	11	0.01
SAMBA Session Service	42808	450.39
NTP	42227	6.12
POP3-PLAIN	26	189.25
POP3-TLS	101	147.68
RTMP	378	2353.67
SMTP-PLAIN	67	62.27
SMTP-TLS	52	3.37
SOCKSv5	1927	898.31
SSH	38961	844.87
Webdav	57	59.91

edge would allow, for example, the end user to adjust the choice of the DPI technique according to the desired level of classification.

4 Dataset

Our basic dataset (without truncated packets or flows) contains 767 690 flows, which account for 53.31 GB of pure packet data. The application name was present for 759 720 flows (98.96 % of all the flows), which account for 51.93 GB (97.41 %) of the data volume. The remaining flows are unlabeled due to their short lifetime (usually below 1 s), which made VBS incapable to reliably establish the corresponding sockets. The application protocols together with the number of flows and the data volume are shown in Table 2, while the applications in Table 3 and the web services in Table 4. The data volume is presented here only for an overview – the rest of the paper uses only the number of flows as the reference value.

We are going to publish our basic labeled dataset with full packet payloads on our website [44]. Therefore, it can be used by the research community as a reference benchmark for the validation of network traffic classifiers.

5 Results

This section provides an overview of the classification results of the different types of traffic by each of the classifiers. The evaluation was performed on 3 datasets: a normal

Table 3: Applications in the Dataset

Application	Number of Flows	Number of Megabytes
4Shared	144	13.39
America's Army	350	61.15
BitTorrent clients (encrypted)	96399	3313.98
BitTorrent clients (non-encrypted)	261527	6779.95
Dropbox	93	128.66
eDonkey clients (obfuscated)	12835	8178.74
eDonkey clients (non-obfuscated)	13852	8480.48
Freenet	135	538.28
FTP clients (active)	126	341.17
FTP clients (passive)	122	270.46
iTunes	235	75.4
League of Legends	23	124.14
Pando Media Booster	13453	13.3
PPLive	1510	83.86
PPStream	1141	390.4
RDP clients	153837	13257.65
Skype (all)	2177	102.99
Skype (audio)	7	4.85
Skype (file transfer)	6	25.74
Skype (video)	7	41.16
Sopcast	424	109.34
Spotify	178	195.15
Steam	1205	255.84
TOR	185	47.14
World of Warcraft	22	1.98

Table 4: Web Services in the Dataset

Web Service	Number of Flows	Number of Megabytes
4Shared	98	68.42
Amazon	602	51.02
Apple	477	90.22
Ask	171	1.86
Bing	456	36.84
Blogspot	235	10.53
CNN	247	3.66
Craigslist	179	4.09
Cyworld	332	13.06
Doubleclick	1989	11.24
eBay	281	8.31
Facebook	6953	747.35
Go.com	335	25.83
Google	6541	532.54
Instagram	9	0.22
Justin.tv	2326	126.33
LinkedIn	62	2.14
Mediafire	472	27.99
MSN	928	23.22
Myspace	2	2.54
Pinterest	189	3.64
Putlocker	103	71.92
QQ.com	753	10.46
Taobao	387	24.29
The Huffington Post	71	21.19
Tumblr	403	52.56
Twitter	1138	13.67
Vimeo	131	204.45
Vk.com	343	9.59
Wikipedia	6092	521.95
Windows Live	26	0.16
Wordpress	169	33.31
Yahoo	17373	937.07
YouTube	2534	1891.79

set, a set with truncated packets, and a set with truncated flows. This section presents a description of the most interesting results, but a discussion is later presented in Section 6 with the conclusions that can be drawn from them. The following subsections present the results for the normal dataset, while the results for the sets with truncated packets or flows are discussed separately. All the accuracy results are given in terms of flows. The methodology used to compute the accuracy is shown in Section 3.3. The interested reader may find the complete confusion matrix in the technical report [39].

5.1 Application Protocols

The evaluation of the classification of application protocols is shown in Table 5, Table 6, and Table 7. The columns present the percentage of correctly and wrongly classified as well as unclassified flows belonging to various application protocols by each classifier.

An important performance parameter of DPI-based techniques is the completeness of their results (i.e., number of applications they can classify). This section evaluates 17 different application protocols. As shown in the tables, none of the techniques is able to classify all of them. Among the different techniques studied, nDPI and Libprotoident are the most complete, classifying 15 out of 17. At the far end, L7-filter only classifies 9 of 17.

Another important aspect of DPI techniques is their ratio of false positives (i.e., incorrect classifications). Usually techniques leave the non-recognized flows as unclassified, trying to decrease the number of false positives. Even though, both versions of L7-filter are characterized for producing a high number of incorrect classifications (e.g., L7-filter-all classifies 85.79% of HTTP traffic as Finger). Regarding the specific classifications, most of traditional application protocols (i.e., DNS, HTTP, IMAP-STARTTLS, POP3-PLAIN, SMTP-PLAIN and SSH) are generally well detected by all the techniques (e.g., accuracy between 70.92% and 100%). Unexpectedly, Libprotoident is the only classifier able to identify all the tested encrypted protocols. Regardless of the classifier, the undetected encrypted traffic is usually identified as regular SSL. An interesting case is presented by the classification of RTMP. Only nDPI and Libprotoident are able to properly classify it. PACE and OpenDPI classify this traffic as Flash. Although both traffics are usually related, the classification as Flash cannot be considered as being correct, as Flash is only a content container. Flash content (audio, video or any other binary file) can be transported using various applications protocols (e.g., HTTP, RTMP) or even different transport protocols (both TCP and UDP).

5.2 Applications

The second level of classification studies the *application* that uses its proprietary application-level protocols (e.g., BitTorrent, Skype). The evaluation of the classification of various *applications* is shown in Table 8, Table 9, Table 10, and Table 11. The columns present

Table 5: Evaluation of Application Protocols – Part 1

Protocol	Classifier	Correct [%]	Wrong [%]	Unclassified [%]
DNS	PACE	99.95	0.00	0.05
	OpenDPI	99.99	0.00	0.01
	L7-filter-all	99.62	0.05	0.33
	L7-filter-com	99.62	0.02	0.36
	nDPI	100.00	0.00	0.00
	Libprotoident	99.96	0.00	0.04
	NBAR	99.99	0.00	0.01
HTTP	PACE	70.92	0.63	28.45
	OpenDPI	95.68	0.59	3.73
	L7-filter-all	3.58	96.04	0.38
	L7-filter-com	35.25	10.28	54.47
	nDPI	17.25	0.83	81.92
	Libprotoident	99.80	0.07	0.13
	NBAR	99.04	0.17	0.79
ICMP	PACE	100.00	0.00	0.00
	OpenDPI	100.00	0.00	0.00
	L7-filter-all	0.00	0.00	100.00
	L7-filter-com	0.00	0.00	100.00
	nDPI	100.00	0.00	0.00
	Libprotoident	100.00	0.00	0.00
	NBAR	100.00	0.00	0.00
IMAP STARTTLS	PACE	100.00	0.00	0.00
	OpenDPI	100.00	0.00	0.00
	L7-filter-all	100.00	0.00	0.00
	L7-filter-com	100.00	0.00	0.00
	nDPI	100.00	0.00	0.00
	Libprotoident	100.00	0.00	0.00
	NBAR	100.00	0.00	0.00
IMAP TLS	PACE	0.00	0.00	100.00
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	0.00	100.00
	L7-filter-com	0.00	0.00	100.00
	nDPI	0.00	0.00	100.00
	Libprotoident	100.00	0.00	0.00
	NBAR	100.00	0.00	0.00
NETBIOS Name Service	PACE	99.96	0.00	0.04
	OpenDPI	98.51	0.00	1.49
	L7-filter-all	0.00	5.63	94.37
	L7-filter-com	0.00	9.15	90.85
	nDPI	99.97	0.00	0.03
	Libprotoident	0.04	4.94	95.02
	NBAR	100.00	0.00	0.00

Table 6: Evaluation of Application Protocols – Part 2

Protocol	Classifier	Correct [%]	Wrong [%]	Unclassified [%]
	PACE	100.00	0.00	0.00
	OpenDPI	100.00	0.00	0.00
NETBIOS	L7-filter-all	9.09	0.00	90.91
Session	L7-filter-com	9.09	0.00	90.91
Service	nDPI	100.00	0.00	0.00
	Libprotoident	100.00	0.00	0.00
	NBAR	100.00	0.00	0.00
	PACE	100.00	0.00	0.00
	OpenDPI	100.00	0.00	0.00
SAMBA	L7-filter-all	100.00	0.00	0.00
Session	L7-filter-com	100.00	0.00	0.00
Service	nDPI	100.00	0.00	0.00
	Libprotoident	100.00	0.00	0.00
	NBAR	0.00	0.00	100.00
	PACE	100.00	0.00	0.00
	OpenDPI	100.00	0.00	0.00
	L7-filter-all	99.86	0.14	0.00
NTP	L7-filter-com	99.86	0.13	0.01
	nDPI	100.00	0.00	0.00
	Libprotoident	100.00	0.00	0.00
	NBAR	0.00	0.00	100.00
	PACE	100.00	0.00	0.00
	OpenDPI	100.00	0.00	0.00
	L7-filter-all	100.00	0.00	0.00
POP3	L7-filter-com	100.00	0.00	0.00
PLAIN	nDPI	100.00	0.00	0.00
	Libprotoident	100.00	0.00	0.00
	NBAR	100.00	0.00	0.00
	PACE	0.00	0.00	100.00
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	5.93	94.06
POP3	L7-filter-com	0.00	0.99	99.01
TLS	nDPI	88.12	0.00	11.88
	Libprotoident	100.00	0.00	0.00
	NBAR	100.00	0.00	0.00
	PACE	0.00	0.00	100.00
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	23.54	76.46
RTMP	L7-filter-com	0.00	23.54	76.46
	nDPI	70.90	15.87	13.23
	Libprotoident	86.51	0.26	13.23
	NBAR	0.00	0.26	99.74

Table 7: Evaluation of Application Protocols – Part 3

Protocol	Classifier	Correct [%]	Wrong [%]	Unclassified [%]
SMTP PLAIN	PACE	100.00	0.00	0.00
	OpenDPI	100.00	0.00	0.00
	L7-filter-all	100.00	0.00	0.00
	L7-filter-com	100.00	0.00	0.00
	nDPI	100.00	0.00	0.00
	Libprotoident	100.00	0.00	0.00
	NBAR	100.00	0.00	0.00
SMTP TLS	PACE	0.00	0.00	100.00
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	0.00	100.00
	L7-filter-com	0.00	0.00	100.00
	nDPI	3.85	0.00	96.15
	Libprotoident	100.00	0.00	0.00
	NBAR	0.00	0.00	100.00
SOCKSv5	PACE	78.26	0.00	21.74
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	100.00	0.00
	L7-filter-com	0.00	100.00	0.00
	nDPI	92.99	0.00	7.01
	Libprotoident	100.00	0.00	0.00
	NBAR	0.00	0.00	100.00
SSH	PACE	93.98	0.51	5.51
	OpenDPI	93.98	0.12	5.90
	L7-filter-all	94.19	0.36	5.45
	L7-filter-com	94.19	0.12	5.69
	nDPI	93.98	0.80	5.22
	Libprotoident	94.19	0.36	5.45
	NBAR	93.71	0.64	5.65
Webdav	PACE	3.51	0.00	96.49
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	7.02	92.98
	L7-filter-com	0.00	7.02	92.98
	nDPI	0.00	0.00	100.00
	Libprotoident	0.00	0.00	100.00
	NBAR	0.00	0.00	100.00

the percentage of correctly and wrongly classified as well as unclassified flows belonging to various *applications* by each of the classifier.

At *application* level the most complete technique is PACE, classifying 20 out of the 22 evaluated *applications*, followed by nDPI (17) and Libprotoident (14). Again, L7-filter is among the worst techniques (8), but it is overcame by NBAR that classifies only 4 *applications*.

Regarding the false positive ratio, the lowest percentage of misclassified flows were obtained from PACE and OpenDPI. Contrary, the highest ratio of misclassified flows is again obtained from the classifications by both versions of L7-filter (e.g., 97% of America's Army traffic is classified as Skype and RTP).

As shown in the tables, the classification on *application* level presents more problems than on *application protocol* level. It is particularly striking that almost no classifier is able to completely classify all the flows (i.e., 100%) from a specific *application*. This can be derived from the fact that usually *applications* use different internal operations that produce different traffic. Therefore, techniques need a specific pattern for every type of operation. For instance, the accuracy with Skype is always lower than 100% because none of the techniques is able to classify neither Skype file transfers nor videos. Among the different studied techniques, PACE is the most accurate followed by nDPI and Libprotoident. Surprisingly, PACE presents severe problems with a traditional *application* as FTP, almost non classifying all its traffic. Another interesting observations extracted from the results are shown below:

- L7-filter, the most unreliable on this level, usually misclassifies the flows as Skype and Finger. However, around 1/3 of the Skype flows are misclassified by it as RTP, Finger, eDonkey, or NTP.
- The authors of traffic classifiers focus on popular applications, which either generate heavy data volume, or are critical regarding QoS requirements. Non-encrypted BitTorrent flows and Skype flows are the only groups of applications that are generally well detected by all the classifiers.
- America's Army game is not classified by any tool. The few correct classifications obtained by nDPI are due to the recognition of some flows originated by the TeamSpeak client integrated with the game.

5.3 Web Services

The last level studied evaluates many different *web services*. Because of clarity and understanding, we do not present the results as a table but as a summary of the important outcomes.

The results with *web services* follow the outcomes obtained on previous levels. PACE is the most complete and accurate technique. The bad results of the rest of techniques

Table 8: Evaluation of Applications – Part 1

Application	Classifier	Correct [%]	Wrong [%]	Unclassified [%]
4Shared	PACE	27.08	0.00	72.92
	OpenDPI	27.08	0.00	72.92
	L7-filter-all	0.00	1.39	98.61
	L7-filter-com	0.00	0.00	100.00
	nDPI	0.00	0.00	100.00
	Libprotoident	0.00	0.00	100.00
	NBAR	0.00	0.00	100.00
America's Army	PACE	0.00	0.00	100.00
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	97.71	2.29
	L7-filter-com	0.00	97.43	2.57
	nDPI	4.00	0.00	96.00
	Libprotoident	0.00	89.14	10.86
	NBAR	0.00	72.00	28.00
BitTorrent clients (encrypted)	PACE	78.68	0.05	21.27
	OpenDPI	0.27	0.00	99.73
	L7-filter-all	40.54	10.17	49.29
	L7-filter-com	40.62	7.30	52.08
	nDPI	54.41	0.18	45.41
	Libprotoident	60.31	0.02	39.67
	NBAR	1.29	0.63	98.08
BitTorrent clients (non-encrypted)	PACE	99.87	0.00	0.13
	OpenDPI	80.61	0.00	19.39
	L7-filter-all	94.56	0.49	4.95
	L7-filter-com	94.60	0.42	4.98
	nDPI	99.41	0.02	0.57
	Libprotoident	99.30	0.00	0.70
	NBAR	77.84	0.36	21.80
Dropbox	PACE	94.62	0.00	5.38
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	0.00	100.00
	L7-filter-com	0.00	0.00	100.00
	nDPI	98.92	0.00	1.08
	Libprotoident	0.00	0.00	100.00
	NBAR	0.00	0.00	100.00
eDonkey clients (obfuscated)	PACE	36.06	7.26	56.68
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	11.64	16.59	71.77
	L7-filter-com	11.64	11.09	77.27
	nDPI	11.04	2.67	86.29
	Libprotoident	11.47	0.00	88.53
	NBAR	0.00	15.93	84.07
eDonkey clients (non-obfuscated)	PACE	16.50	3.74	79.76
	OpenDPI	3.98	0.30	95.72
	L7-filter-all	17.97	16.32	65.71
	L7-filter-com	17.99	10.79	71.22
	nDPI	15.57	2.28	82.23
	Libprotoident	17.86	0.31	81.83
	NBAR	2.05	11.19	86.76

Table 9: Evaluation of Applications – Part 2

Application	Classifier	Correct [%]	Wrong [%]	Unclassified [%]
	PACE	79.26	0.00	20.74
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	20.00	80.00
Freenet	L7-filter-com	0.00	14.07	85.93
	nDPI	0.00	3.70	96.30
	Libprotoident	0.00	0.00	100.00
	NBAR	0.00	15.56	84.44
	PACE	5.56	0.00	94.44
	OpenDPI	97.62	0.00	2.38
FTP	L7-filter-all	5.56	92.06	2.38
clients	L7-filter-com	5.56	90.47	3.97
(active)	nDPI	98.41	0.00	1.59
	Libprotoident	100.00	0.00	0.00
	NBAR	50.79	0.00	49.21
	PACE	4.92	0.00	95.08
	OpenDPI	67.21	0.00	32.79
FTP	L7-filter-all	4.92	76.23	23.77
clients	L7-filter-com	4.92	73.77	26.23
(passive)	nDPI	72.95	0.00	27.05
	Libprotoident	73.77	22.95	32.80
	NBAR	50.00	0.00	50.00
	PACE	77.45	0.00	22.55
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	63.83	6.81	29.36
iTunes	L7-filter-com	63.83	0.00	36.17
	nDPI	13.19	0.00	86.81
	Libprotoident	0.00	0.00	100.00
	NBAR	0.00	0.00	100.00
	PACE	0.00	13.04	86.96
	OpenDPI	0.00	0.00	100.00
League	L7-filter-all	0.00	69.57	30.43
of	L7-filter-com	0.00	4.35	95.65
Legends	nDPI	0.00	13.04	86.96
	Libprotoident	0.00	4.35	95.65
	NBAR	0.00	0.00	100.00
	PACE	99.45	0.39	0.16
	OpenDPI	99.23	0.54	0.23
Pando	L7-filter-all	0.00	0.74	99.26
Media	L7-filter-com	0.00	0.55	99.45
Booster	nDPI	99.26	0.63	0.11
	Libprotoident	99.26	0.41	0.33
	NBAR	0.00	0.36	99.64
	PACE	88.21	0.00	11.79
	OpenDPI	0.07	0.13	99.80
	L7-filter-all	0.00	56.03	43.97
PPLive	L7-filter-com	0.00	17.15	82.85
	nDPI	43.91	1.05	55.04
	Libprotoident	43.91	1.05	55.04
	NBAR	0.00	0.40	99.60

Table 10: Evaluation of Applications – Part 3

Application	Classifier	Correct [%]	Wrong [%]	Unclassified [%]
PPStream	PACE	79.32	0.00	20.68
	OpenDPI	0.79	0.00	99.21
	L7-filter-all	0.00	38.39	61.61
	L7-filter-com	0.00	15.07	84.93
	nDPI	0.53	0.26	99.21
	Libprotoident	0.96	0.00	99.04
	NBAR	0.00	5.26	94.74
RDP clients	PACE	99.69	0.00	0.31
	OpenDPI	99.70	0.00	0.30
	L7-filter-all	0.00	92.25	7.75
	L7-filter-com	0.00	92.25	7.75
	nDPI	99.69	0.02	0.29
	Libprotoident	99.66	0.01	0.33
	NBAR	0.00	0.67	99.33
Skype (all)	PACE	83.51	5.05	11.44
	OpenDPI	38.49	0.32	61.19
	L7-filter-all	59.21	31.70	9.09
	L7-filter-com	62.52	24.67	12.81
	nDPI	99.82	0.00	0.18
	Libprotoident	88.75	0.00	11.25
	NBAR	70.37	3.40	26.23
Skype (audio)	PACE	100.00	0.00	0.00
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	85.71	14.29	0.00
	L7-filter-com	100.00	0.00	0.00
	nDPI	0.00	0.00	100.00
	Libprotoident	0.00	0.00	100.00
	NBAR	0.00	0.00	100.00
Skype (file transfer)	PACE	0.00	100.00	0.00
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	100.00	0.00
	L7-filter-com	0.00	100.00	0.00
	nDPI	0.00	0.00	100.00
	Libprotoident	0.00	0.00	100.00
	NBAR	0.00	0.00	100.00
Skype (video)	PACE	0.00	100.00	0.00
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	100.00	0.00
	L7-filter-com	0.00	100.00	0.00
	nDPI	0.00	0.00	100.00
	Libprotoident	0.00	0.00	100.00
	NBAR	0.00	0.00	100.00
Sopcast	PACE	66.27	3.07	30.66
	OpenDPI	66.27	2.59	31.14
	L7-filter-all	0.00	99.06	0.94
	L7-filter-com	0.00	74.76	25.24
	nDPI	63.68	1.18	35.14
	Libprotoident	46.70	0.24	53.06
	NBAR	0.00	0.00	100.00

Table 11: Evaluation of Applications – Part 4

Application	Classifier	Correct [%]	Wrong [%]	Unclassified [%]
Spotify	PACE	37.64	2.25	60.11
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	43.26	56.74
	L7-filter-com	0.00	10.11	89.89
	nDPI	0.56	3.93	95.51
	Libprotoident	0.56	0.00	99.44
	NBAR	0.00	0.56	99.44
Steam	PACE	55.19	0.75	44.06
	OpenDPI	0.33	0.00	99.67
	L7-filter-all	0.00	65.89	34.11
	L7-filter-com	0.00	4.73	95.27
	nDPI	76.02	0.42	23.56
	Libprotoident	75.85	0.00	24.15
	NBAR	0.00	0.58	99.42
TOR	PACE	85.95	0.00	14.05
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	0.00	100.00
	L7-filter-com	0.00	0.00	100.00
	nDPI	33.51	0.00	66.49
	Libprotoident	33.51	0.00	66.49
	NBAR	0.00	2.16	97.84
World of Warcraft	PACE	27.27	0.00	72.73
	OpenDPI	0.00	0.00	100.00
	L7-filter-all	0.00	86.36	13.64
	L7-filter-com	0.00	22.73	77.27
	nDPI	13.64	13.64	72.72
	Libprotoident	13.64	0.00	86.36
	NBAR	0.00	0.00	100.00

are mainly due to a not enough specific classification (e.g., Facebook traffic classified as HTTP).

PACE recognizes 4Shared (84.69 %), Amazon (58.97 %), Apple (0.84 %), Blogspot (3.83 %), eBay (67.97 %), Facebook (80.79 %), Google (10.79 %), Instagram (88.89 %), Linkedin (77.42 %), Mediafire (30.30 %), Myspace (100 %), QQ.com (32.14 %), Twitter (71.18 %), Windows Live (96.15 %), Yahoo (54.70 %), and YouTube (81.97 %). PACE does not have problems with recognizing SSL flows belonging to these services, which means that PACE must use other techniques than just looking directly into the packets to associate the flows with the particular services, probably by analyzing the server certificates.

The commercial tool clearly overcomes its open-source version OpenDPI that recognizes only Direct Download websites: 4Shared (83.67 %) and MediaFire (30.30 %).

L7-filter recognizes only Apple (0.42 %). Furthermore, L7-filter (especially L7-filter-all) is characterized by a very high number of misclassified flows belonging to web services (usually 80–99 %). The flows are recognized in a vast majority as Finger and Skype.

nDPI recognizes Amazon (83.89 %), Apple (74.63 %), Blogspot (4.68 %), Doubleclick (85.92 %), eBay (72.24 %), Facebook (80.14 %), Google (82.39 %), Yahoo (83.16 %), Wikipedia (68.96 %), and YouTube (82.16 %) being the second best technique on this level.

Unlike on previous levels, Libprotoident recognizes only the Yahoo (2.36 %) *web service*. This result is understandable given that Libprotoident only uses the first 4 bytes of packet payload to classify a flow, making considerably more difficult a specific classification as web service.

The worst technique on this level is NBAR that does not recognize any web services.

5.4 Impact of Packet Truncation

An important characteristic of each DPI tool is the amount of information needed from each packet to identify the traffic. That significantly influences the classification speed and the resources needed. Furthermore, many traffic traces are published with payload truncated up to a certain number of bytes per packet for privacy reasons. As mentioned before, Libprotoident is the only tool, which is advertised to use the particular extent of the examined packets, namely first 4 bytes. Therefore, in order to discover the internal properties of each tool, we decided to test the impact of packet truncation. This subsection presents the differences between the classification results for the normal dataset and the dataset with truncated Ethernet frames to the first 70 B.

Truncation of packets has a considerable impact on the classification of most *application protocols* by all tools except Libprotoident and NBAR, which tend to maintain their normal accuracy. This suggests that NBAR can be somehow implemented as Libprotoident to classify *application protocols* while the rest of techniques base their

classification on the complete flow. L7-filter is not able to detect DNS traffic on this set, while all the other classifiers present the accuracy of over 99 %. Unexpectedly, NBAR cannot detect NTP on the normal set, while it detects it 100 % correctly on the set with truncated packets. We can not present a verifiable reason of this result given that NBAR is not an open-source tool.

At *application* level, only Libprotoident is able to keep its normal accuracy whereas the rest of techniques considerably decreases their accuracies.

Regarding the *web services* level, only nDPI is able to recognize some *web services* in this set. Exceptionally, the detection rate is almost the same as for the normal set. Other classifiers tend to leave such traffic as *unknown*.

5.5 Impact of Flow Truncation

Another major concern is how many packets are needed in order to classify each flow. That depends on the classification tool as well as on the application or protocol, which we want to identify. However, the documentation of the traffic classifiers do not cover these issues, although they are very important for conserving disk space while publishing data traces used to test the tools. Therefore, we decided to study the impact of flow truncation. This subsection presents the differences between the classification results for the normal dataset and the dataset with truncated flows to the first 10 packets.

Truncation of flows does not have any noticeable impact on the classification of *application protocols*. This result suggests that the classification of *application protocols* relies on patterns or signatures extracted from the first packets of the flows.

Similar behavior is obtained on *application* level. However, in this case the impact on the classification of *applications* is noticeable – the detection rate decreases. The only exception is Libprotoident, which provides the same results as for the normal dataset. Therefore, this insinuate that the classification of some *applications* probably rely on techniques based on statistics (e.g., Machine Learning). FTP in the active mode is a very interesting case, as Libprotoident maintains its 100 % accuracy, while the accuracy of the other classifiers drops to 5.56 %. An strange case is presented with Plain eDonkey traffic, as the best classification accuracy (45.28 %) we obtained by using PACE on the set with truncated flows, while the accuracy on the normal set was only 16.50 %.

The percentage of correctly classified *web services* is usually the same or nearly the same as for the normal set.

6 Discussion

This section extracts the outcomes from the results obtained during the performance comparison. Also, we discuss the limitations of our study.

As it is shown in the previous section, *PACE* is the best classifier for most of the studied classification groups. This high ranking is due to the ability of providing the

results on various levels, as for example *HTTP:generic:facebook*. Other classifiers do not offer this ability at all and only one chosen level is given, so, for example, they do not offer the possibility to account the HTTP or SSL traffic, while they recognize the web service of the transported content. However, PACE also is not totally consistent in that matter. Facebook videos (which we observed as transported by HTTP) were detected as, for example, *FLASH:no_subprotocols:facebook*, while the live video streams from Justin.tv using RTMP, were classified as *FLASH:no_subprotocols:not_detected*. So, we do not have the knowledge from the results obtained from the classifier which application protocol was used (HTTP, RTMP, or other), because the content container level is returned instead. Ideally, the DPI techniques should provide results on all the possible levels, as *HTTP:FLASH:VIDEO:YOUTUBE*, so that kind of consistent accounting could be performed. However, PACE is a commercial tool not accessible for all the research community. Among the available open-source tools, *nDPI* and *Libprotoident* reveal as the most reliable solutions. Surprisingly for us, Libprotoident achieves very good results without giving a noticeable number of false classifications by using the first four bytes of payload for each direction. On the other hand, *L7-filter* and *NBAR* perform poorly in classifying the traffic from our dataset.

We did not observe large differences between the classifications performed on the normal dataset and the set with truncated flows to maximum 10 packets. The set with truncated packets is usually much worse classified than the other sets by all tools except Libprotoident, which maintains the same accuracy. We found that our modified version of L7-filter-com provides overall better results than the default L7-filter-all by increased number of correct classifications and greatly reduced rate of misclassifications (especially, regarding the web services).

Nonetheless, the previous conclusions are obviously tied to our dataset. Although we have tried our best to emulate the real behavior of the users, many applications, behaviors and configurations are not represented on it. Because of that it has some limitations that we discuss next:

- In our study we have evaluated 17 well-known application protocols, 19 applications (including 4 in various configurations), and 34 web services. The results obtained from the different classifiers are directly related to those groups. Thus, the introduction of different groups could arise different outcomes.

- The traffic generated for building the dataset, although has been manually and realistically created, is artificial. The backbone traffic would carry different behaviors of the groups that are not fully represented in our dataset (e.g., P2P clients running on port 80). The performance of the tools studied might not be directly extrapolated from the current results. However, the artificially created traffic allowed us to publish the dataset with full packet payloads.

- The poor performance of NBAR and L7-filter might be affected by the characteristics of our dataset. Thus, the reliability of previous works based on them is

not called into question. Different configurations [10, 42, 45] and different or older classification groups would probably produce different results.

- The classification levels have considerable impact on the results. For instance, classifying Facebook, Google or Twitter is currently not possible by Libprotoident, however it is possible by nDPI and PACE.
- The amount of data available would also have impacted on the performance. The study presented in this paper is performed with full payload packets. However, in other works the traces are usually collected with a few bytes of data [13, 46, 47] (e.g., 96 bytes) in order to avoid packet loss, disk space, and privacy issues. For this scenario, it seems that Libprotoident is a more suitable solution, giving it only uses the first 4 bytes of every packet.
- The nature, distribution, and heterogeneity of the traffic would also impact the performance. The amount of classes detected by PACE is considerably bigger than detected by the rest of the classifiers, which makes PACE more suitable for heterogeneous scenarios. Also, PACE and nDPI are able to classify traffic in asymmetric scenarios.

7 Related Work

This section reviews the literature related to the comparison of DPI tools. The OpenDPI tool amounts for most of the publications [6, 7, 10, 48, 49]. According to [6], the test performed by the European Networking Tester Center (EANTC) in 2009 resulted in 99 % of detection and accuracy for popular P2P protocols by OpenDPI. The big amount of flows marked as *unknown* by OpenDPI was confirmed in [48], where the authors made an effort to calculate various parameters for traffic originated from different applications: number of flows, data volume, flow sizes, number of concurrent flows, and inter-arrival times. The study was based on 3.297 TB of data collected during 14 days from an access network with connected around 600 households. 80.1 % of the flows, amounting for 64 % of the traffic volume, were marked as *unknown* by OpenDPI.

In [6], the authors study the impact of per-packet payload sampling (i.e., packet truncation) and per-flow packet sampling (i.e., collect only the first packets of a flow) on the performance of OpenDPI. The results show that OpenDPI is able to keep the accuracy higher than 90-99% with only the first 4-10 packets of a flow. The impact by the per-packet payload sampling is considerably higher. Their results use as ground-truth the dataset labeled by OpenDPI with no sampling. Thus, the actual classification of the dataset is unknown and no possible comparison with our work can be done.

Similar work, performed by the same authors, is described in [7]. The goal was to find out what is the suggested number of packets from each flow, which needs to be inspected by OpenDPI in order to achieve good accuracy, while maintaining a low

computational cost. The focus was on Peer-to-Peer (P2P) protocols. The test was performed on a 3 GB randomly selected subset of flows from the data collected at an access link of an institution over 3 days. The authors found that inspecting only 10 packets from each flow lowered the classification abilities of P2P flows by OpenDPI by just 0.85 % comparing to the classification of full flows, while saving more than 9 % of time.

In [10], the authors tested the accuracy of L7-filter and OpenDPI, and they also built their own version of L7-filter with enhanced abilities of classification of the UDP traffic. The data used in the experiment were collected by Wireshark, while the applications were running in the background. The data were split into 27 traces, each for one application, where all the applications were supported by both L7-filter and OpenDPI. Other flows were removed from the dataset. However, they do not explain how they validate the process of the isolation of the different applications. The obtained precision was 100 % in all the cases (none of the classification tools gave a false positive), while the recall deviated from 67 % for the standard L7-filter, through 74 % for their own implementation of L7-filter, and 87 % for OpenDPI.

Fukuda compared in [47] the performance of L7-filter and OpenDPI on the backbone traffic. The dataset used is characterized as being in majority asymmetric and containing the packets truncated after 96 Bytes. The ground-truth is labeled using a port-based technique and then the three DPI-based techniques are compared. The results show that the DPI-based techniques are only able to classify 40-60% of the traffic in this scenario.

In [8], the developers of Libprotoident evaluated the accuracy of the classification of this tool and compared the results with OpenDPI, Nmap, and L7-filter. The ground-truth was established by PACE, so only the flows recognized by PACE were taken into account during the experiment. The accuracy was tested on two datasets: one taken from the Auckland university network, and one from an Internet Service Provider (ISP). On the first dataset, Libprotoident had the lowest error rate of less than 1 % (OpenDPI: 1.5 %, L7-filter: 12.3 %, Nmap: 48 %.). On the second dataset, Libprotoident achieved the error rate of 13.7 %, while OpenDPI 23.3 %, L7-filter 22 %, and Nmap 68.9 %. The authors claim that Libprotoident identified 65 % of BitTorrent traffic and nearly 100 % of HTTP, SMTP, and SSL. Same authors also compared in [9] four open-source DPI-based tools (i.e., nDPI, Tstat, Libprotoident, and L7-filter). Similarly to us, they artificially built a labeled dataset using a complicate mix of filters in an isolated host. Unlike us, their trace is not available to the community so no further comparison is possible. However, their results confirms some of the findings of our paper presenting nDPI and Libprotoident as the most accurate open-source DPI-based tools.

Another lightweight packet inspection approach was proposed in [50]. The authors developed PortLoad, which was designed to be characterized by the speed of port-based classifiers, while maintaining the accuracy of DPI tools. The authors showed that almost all the matching strings start (99.98 %) and finish (90.77 %) in the first 32

bytes of payload. Only the first packet in each direction is processed. PortLoad was compared against L7-filter and the port-based approach. The experimental evaluation showed that the processing time is cut down by more than 97 % comparing to L7-filter, while the accuracy was assessed to be 74 %.

To the best of our knowledge, there are no accessible research studies or reports about the accuracy of NBAR. However, an experiment was made to assess how big amount of network traffic is classified by NBAR and L7-filter, and how big amount of traffic is left as *unknown* [51]. The authors captured by Wireshark all the packets flowing in a local network of an IT company during 1 hour. From 27 502 observed packets, 12.56 % were reported as unknown by NBAR, and 30.44 % were reported as unknown by L7-filter.

A very comprehensive review of different methods for traffic classification was made in 2013 by Silvio Valenti et al. [52]. The authors refer to 68 different positions in the literature and cover the topic from the basis to more advanced topics, mostly dealing with Machine Learning Algorithms (MLAs).

8 Conclusions

This paper presents a reliable evaluation of the accuracy of some of the most well-known DPI-based network traffic classifiers. We compared the precision of six tools (i.e., *PACE*, *OpenDPI*, *L7-filter*, *nDPI*, *Libprotoident*, and *NBAR*), which are usually used for traffic classification. The results obtained in Section 5 and further discussed in Section 6 show that *PACE* is, on our dataset, the most reliable solution for traffic classification. Among the open-source tools, *nDPI* and *Libprotoident* present the best results. The choice between them would depend on the scenario or the level on which we would like to obtain the results. On the other hand, *NBAR* and *L7-filter* present several inaccuracies that make them not recommendable for network traffic classification in their current form.

In order to make the study trustworthy, we created a dataset using VBS [20]. This tool associates the name of the process to each flow making its labeling totally reliable. The dataset of more than 750 K flows contains traffic from popular applications. Also, this dataset allows the validation of different techniques on different levels (i.e., application protocol, application, and web service). The total amount of data properly labeled is 51.93 GB. Furthermore, and more important, we released to the research community this dataset with full payload, so it can be used as a common reference for the comparison and validation of network traffic classifiers.

Although this study is complete, the continuous evolution of the network applications and the DPI-based techniques allows a periodical updated of the evaluation. For instance, this evaluation can be updated by adding new applications and web services to the dataset (e.g., Netflix) and by introducing new classification tools to the study (e.g., NBAR2 or Tstat). In this paper, we focused on the reliability of the DPI tools, however, a possible line of future work can be related to their deployment for real-time

classification (i.e., scalability and computational cost).

9 Acknowledgments

This research was funded by the Spanish Ministry of Science and Innovation under contract TEC2011-27474 (NOMADS project) and by AGAUR (ref. 2014-SGR-1427). This work was also co-financed (grant no. 8–10100) by Aalborg University, the European Regional Development Fund (ERDF), and Bredbånd Nord A/S.

References

[1] NBAR2 or Next Generation NBAR – Cisco Systems, 2013. [Online]. Available: http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6558/ps6616/qa_c67-697963.html.

[2] PACE | Network Analysis with Layer-7 Deep Packet Inspection, 2013. [Online]. Available: http://www.ipoque.com/en/products/pace.

[3] Qosmos | Deep Packet Inspection and Metadata Engine, 2013. [Online]. Available: http://www.qosmos.com/products/deep-packet-inspection-engine/.

[4] Paloalto Networks, 2013. [Online]. Available: https://www.paloaltonetworks.com/.

[5] Alberto Dainotti, Antonio Pescapè, and Kimberly C. Claffy. Issues and future directions in traffic classification. *IEEE Network*, 26(1):35–40, 2012. DOI: 10.1109/MNET.2012.6135854.

[6] Jawad Khalife, Amjad Hajjar, and Jesús Díaz-Verdejo. Performance of OpenDPI in Identifying Sampled Network Traffic. *Journal of Networks*, 8(1):71–81, January 2013. DOI: 10.4304/jnw.8.1.71-81.

[7] Jawad Khalife, Amjad Hajjar, and Jesús Díaz-Verdejo. On the Performance of OpenDPI in Identifying P2P Truncated Flows. In *AP2PS 2011, The Third International Conference on Advances in P2P Systems*, pages 79–84. IARIA, Lisbon, Portugal, November 2011.

[8] Shane Alcock and Richard Nelson. Libprotoident: Traffic Classification Using Lightweight Packet Inspection. Technical report, University of Waikato, August 2012. Accessible: http://www.wand.net.nz/publications/lpireport.

[9] Alcock, Shane and Nelson, Richard. Measuring the Accuracy of Open-Source Payload-Based Traffic Classifiers Using Popular Internet Applications. In *IEEE*

Workshop on Network Measurements (WNM), the 38th IEEE Conference on Local Computer Networks (LCN). IEEE, Sydney, Australia, October 2013.

[10] Chaofan Shen and Leijun Huang. On detection accuracy of L7-filter and OpenDPI. In *2012 Third International Conference on Networking and Distributed Computing (ICNDC)*, pages 119–123. IEEE, Hangzhou, China, October 2012. DOI: 10.1109/ICNDC.2012.36.

[11] CAIDA Internet Data – Passive Data Sources, 2012. [Online]. Available: http://www.caida.org/data/passive/.

[12] CAIDA Internet Data – Internet Measurement Data Catalog (IMDC), 2012. [Online]. Available: http://www.caida.org/projects/trends/imdc/.

[13] MAWI Working Group Traffic Archive, 2013. [Online]. Available: http://mawi.wide.ad.jp/mawi/.

[14] CRAWDAD, 2013. [Online]. Available: http://crawdad.cs.dartmouth.edu/.

[15] WITS: Waikato Internet Traffic Storage, 2013. [Online]. Available: http://www.wand.net.nz/wits/.

[16] Géza Szabó, Dániel Orincsay, Szabolcs Malomsoky, and István Szabó. On the Validation of Traffic Classification Algorithms. In *Proceedings of the 9th International Conference on Passive and Active network Measurement PAM 2008*, pages 72–81. Springer Berlin Heidelberg, Cleveland, Ohio, USA, April 2008. DOI: 10.1007/978-3-540-79232-1_8.

[17] Francesco Gringoli, Luca Salgarelli, Maurizio Dusi, Niccolo Cascarano, Fulvio Risso, and Kimberly C. Claffy. Gt: picking up the truth from the ground for internet traffic. *ACM SIGCOMM Computer Communication Review*, 39(5):12–18, 2009. DOI: 10.1145/1629607.1629610.

[18] Marco Canini, Wei Li, Andrew W. Moore, and Raffaele Bolla. GTVS: Boosting the Collection of Application Traffic Ground Truth. In *Proceedings of the First International Workshop on Traffic Monitoring and Analysis, TMA 2009*, pages 54–63. Springer Berlin Heidelberg, Aachen, Germany, May 2009. DOI: 10.1007/978-3-642-01645-5_7.

[19] Valentín Carela-Español, Tomasz Bujlow, and Pere Barlet-Ros. Is our ground-truth for traffic classification reliable? In *Proceedings of the 15th Passive and Active Measurement Conference (PAM 2014), Proceedings Series: Lecture Notes in Computer Science 8362*, pages 98–108. Springer International Publishing Switzerland, Los Angeles, USA, March 2014. DOI: 10.1007/978-3-319-04918-2_10.

[20] Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for classification of traffic in computer networks. In *Proceedings of the 19th Telecommunications Forum TELFOR 2011*, pages 210–213. IEEE, Belgrade, Serbia, November 2011. DOI: 10.1109/TELFOR.2011.6143528.

[21] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. nDPI: Open-Source High-Speed Deep Packet Inspection. In *To appear in the 10th International Wireless Communications & Mobile Computing Conference 2014 (IWCMC 2014)*, pages ?–? IEEE, Nicosia, Cyprus, August 2014.

[22] De Sensi, Daniele and Danelutto, Marco and Deri, Luca. Dpi over commodity hardware: implementation of a scalable framework using fastflow. Master's thesis, Università di Pisa, Italy, 2012. Accessible: http://etd.adm.unipi.it/t/etd-02042013-101033/.

[23] Michael Ott. Intelligent network based application recognition, November 2005. US Patent 6,961,770.

[24] Application Layer Packet Classifier for Linux, 2009. [Online]. Available: http://l7-filter.sourceforge.net/.

[25] Tomasz Bujlow, Kartheepan Balachandran, Sara Ligaard Nørgaard Hald, Tahir Riaz, and Jens Myrup Pedersen. Volunteer-Based System for research on the Internet traffic. *TELFOR Journal*, 4(1):2–7, September 2012. Accessible: http://journal.telfor.rs/Published/Vol4No1/Vol4No1.aspx.

[26] Volunteer-Based System for Research on the Internet, 2012. [Online]. Available: http://vbsi.sourceforge.net/.

[27] w3schools.com, OS Platform Statistics, 2013. [Online]. Available: http://www.w3schools.com/browsers/browsers_os.asp.

[28] Palo Alto Networks. APPLICATION USAGE AND THREAT REPORT, 2013. [Online]. Available: https://www.paloaltonetworks.com/resources/whitepapers/application-usage-and-threat-report.html.

[29] CNET, P2P & File-Sharing Software, 2013. [Online]. Available: http://download.cnet.com/windows/p2p-file-sharing-software/?tag=nav.

[30] CNET, FTP Software, 2013. [Online]. Available: http://download.cnet.com/windows/ftp-software/?tag=mncol;sort&rpp=30&sort=popularity.

[31] Top 15 Most Popular File Sharing Websites, 2013. [Online]. Available: http://www.ebizmba.com/articles/file-sharing-websites.

[32] Top 15 Most Popular Video Websites | December 2013, 2013. [Online]. Available: http://www.ebizmba.com/articles/video-websites.

[33] Popular on YouTube – YouTube, 2013. [Online]. Available: http://www.youtube.com/charts/videos_views?t=a.

[34] w3schools.com, Browser Statistics, 2013. [Online]. Available: http://www.w3schools.com/browsers/browsers_stats.asp.

[35] Alexa Top 500 Global Sites, 2013. [Online]. Available: http://www.alexa.com/topsites.

[36] Top 15 Most Popular Websites | December 2013, 2013. [Online]. Available: http://www.ebizmba.com/articles/most-popular-websites.

[37] Quantcast – Top Ranking International Websites, 2013. [Online]. Available: https://www.quantcast.com/top-sites.

[38] Top 15 Most Popular Search Engines – eBizMBA, 2013. [Online]. Available: http://www.ebizmba.com/articles/search-engines.

[39] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Extended Independent Comparison of Popular Deep Packet Inspection (DPI) Tools for Traffic Classification. Technical report, Department of Computer Architecture (DAC), Universitat Politècnica de Catalunya (UPC), January 2014. Accessible: https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2014,en.html.

[40] Email Client Market Share and Popularity – October 2013, 2013. [Online]. Available: http://emailclientmarketshare.com/.

[41] DFC Intelligence, 2013. [Online]. Available: http://www.dfcint.com/wp/?p=343.

[42] Valentín Carela-Español, Pere Barlet-Ros, Alberto Cabellos-Aparicio, and Josep Solé-Pareta. Analysis of the impact of sampling on NetFlow traffic classification. *Computer Networks*, 55:1083–1099, 2011. DOI: 10.1016/j.comnet.2010.11.002.

[43] pmacct project: IP accounting iconoclasm, 2013. [Online]. Available: http://www.pmacct.net/.

[44] Traffic classification at the Universitat Politècnica de Catalunya (UPC)., 2014. [Online]. Available: http://www.cba.upc.edu/monitoring/traffic-classification.

[45] Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Quantifying the accuracy of the ground truth associated with Internet traffic traces. *Computer Networks*, 55(5):1158–1167, April 2011. DOI: 10.1016/j.comnet.2010.11.006.

[46] Hyunchul Kim, Kimberly C. Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In *Proceedings of the 2008 ACM CoNEXT conference*, page 8. ACM New York, Madrid, Spain, December 2008. DOI: 10.1145/1544012.1544023.

[47] Kensuke Fukuda. Difficulties of identifying application type in backbone traffic. In *2010 International Conference on Network and Service Management (CNSM)*, pages 358–361. IEEE, Niagara Falls, Ontario, Canada, October 2010. DOI: 10.1109/CNSM.2010.5691234.

[48] Steffen Gebert, Rastin Pries, Daniel Schlosser, and Klaus Heck. Internet access traffic measurement and analysis. In *Proceedings of the 4th international conference on Traffic Monitoring and Analysis (TMA'12)*, pages 29–42. Springer Berlin Heidelberg, Vienna, Austria, March 2012. DOI: 10.1007/978-3-642-28534-9_3.

[49] Péter Megyesi and Sándor Molnár. Finding Typical Internet User Behaviors. In *Proceedings of the 18th EUNICE Conference on Information and Communications Technologies (EUNICE 2012): Information and Communication Technologies*, pages 321–327. Springer Berlin Heidelberg, Budapest, Hungary, August 2012. DOI: 10.1007/978-3-642-32808-4_29.

[50] Giuseppe Aceto, Alberto Dainotti, Walter De Donato, and Antonio Pescapé. PortLoad: taking the best of two worlds in traffic classification. In *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*, pages 1–5. IEEE, San Diego, California, USA, March 2010. DOI: 10.1109/INFCOMW.2010.5466645.

[51] Ryan Goss and Reinhardt Botha. Deep Packet Inspection – Fear of the Unknown. In *Information Security for South Africa (ISSA), 2010*, pages 1–5. IEEE, Sandton, Johannesburg, South Africa, August 2010. DOI: 10.1109/ISSA.2010.5588278.

[52] Silvio Valenti, Dario Rossi, Alberto Dainotti, Antonio Pescapè, Alessandro Finamore, and Marco Mellia. Reviewing Traffic Classification. In *Data Traffic Monitoring and Analysis*, pages 123–147. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-36784-7_6.

Tomasz Bujlow is a Ph.D. Student in the Department of Electronic Systems at Aalborg University in Denmark. He received his Master of Science in Computer Engineering from Silesian University of Technology in Poland in 2008, specializing in Databases, Computer Networks and Computer Systems. Previously, he obtained his Bachelor of Computer Engineering from University of Southern Denmark in 2009, specializing in software engineering and system integration. His research interests include methods for traffic classification in computer networks. He is also a Cisco Certified Network Professional (CCNP) since 2010.

Valentín Carela-Español received a B.Sc. degree in Computer Science from the Universitat Politècnica de Catalunya (UPC) in 2007 and a M.Sc. degree in Computer Architecture, Networks, and Systems from UPC in 2009. He is currently a Ph.D. Student at the Computer Architecture Department at the UPC. His research interests are in the field of traffic analysis and network measurement, focusing on the identification of applications in network traffic.

Pere Barlet-Ros received the M.Sc. and Ph.D. degrees in Computer Science from the Universitat Politècnica de Catalunya (UPC) in 2003 and 2008, respectively. He is currently an Associate Professor with the Computer Architecture Department of UPC and co-founder of Talaia Networks, a University spin-off that develops innovative network monitoring products. His research interests are in the fields of network monitoring, traffic classification, and anomaly detection.

Paper X

nDPI: Open-Source High-Speed Deep Packet Inspection

Luca Deri[ab], Maurizio Martinelli[a], Tomasz Bujlow[c], and Alfredo Cardigliano[b]

[a] *Institute of Informatics and Telematics (IIT), The National Research Council (CNR), Pisa, Italy*
{luca.deri, maurizio.martinelli}@iit.cnr.it

[b] *ntop, Pisa, Italy*
{deri, cardigliano}@ntop.org

[c] *Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark*
tbu@es.aau.dk

This paper is going to appear in the *Proceedings of the 10th International Wireless Communications & Mobile Computing Conference 2014 (IWCMC2014)*. IEEE, Nicosia, Cyprus, August 2014.

Abstract

Network traffic analysis was traditionally limited to packet headers, because the transport protocol and application ports were usually sufficient to identify the application protocol. With the advent of port-independent, peer-to-peer, and encrypted protocols, the task of identifying application protocols became increasingly challenging, thus creating a motivation for creating tools and libraries for network protocol classification. This paper covers the design and implementation of nDPI, an open-source library for protocol classification using both packet header and payload. nDPI was extensively validated in various monitoring projects ranging from Linux kernel protocol classification, to analysis of 10 Gbit/s traffic, reporting both high protocol detection accuracy and efficiency.

Keywords

passive traffic classification, Deep Packet Inspection, network traffic monitoring

1 Introduction

In the early days of the Internet, network traffic protocols were identified by a protocol and port. For instance, SMTP used TCP port 25 while telnet used TCP port 23. This well-know protocol/port association is specified in the */etc/protocols* file, which is a part of every Unix-based operating system. Over the time, with the advent of Remote Procedure Call (RPC), the use of static ports became a problem. Therefore, specific applications such as *rpcbind* and *portmap* were developed to handle dynamic mappings. Historically, application ports up to 1024 identified essential system services, such as email or remote system login and hence require super-user privileges; their port-to-protocol bindings are preserved until today. Ports above 1024 are used for user-defined services and are generally dynamic.

Protocol identification is often not reliable even when a static port is used. A case in point is TCP/80 used for HTTP. Originally, HTTP was created to carry web-related resources such as HTML pages and decorative content. However, its extensibility (in no small part due to its header flexibility and MIME type specification) along with its native integration in web browsers HTTP is now often used to carry non web-related resources. For instance, it is now the de-facto protocol for downloading/uploading files, thus replacing the File Transfer Protocol (FTP), which was designed specifically for that purpose. The pervasive use of HTTP and its native support of firewalls (i.e., a firewall recognizes and validates the protocol header), made HTTP (and its secure counterpart HTTPS) the ideal developer choice when creating a new protocol that has to traverse a firewall without restrictions. Many peer-to-peer protocols and popular applications (e.g., Skype) use HTTP as the last resort when they need to pass through a firewall

in case all other ports are blocked. We created traffic reports from various networks, ranging from academic sites to commercial ISPs, and realized that HTTP is by far the most widely used protocol. This does not mean that users mostly use it for surfing the web. This protocol is extensively used by social networks, geographical maps, and video-streaming services. In other words the equation TCP/80 = web is no longer valid.

The characterization of network protocols is required not only for creating accurate network traffic reports, but increasingly, for overall network security needs. Modern firewalls combine IP/protocol/port based security with selected protocol inspection in order to validate protocols, in particular those based on UDP, e.g., Simple Network Management Protocol (SNMP) and Domain Name System (DNS). VoIP protocols, such as SIP and H.323, are inspected for specific information, e.g., the IP and port where voice and video will flow. Cisco Network-based Application Recognition (NBAR) devices [1], and Palo Alto Networks application-based firewalls [2] pioneered application-protocol based traffic management. Today, these traffic inspection facilities are available on every modern network security device, because the binding port/protocol scheme no longer holds.

The need to increase network traffic visibility created a need for Deep Traffic Inspection (DPI) libraries to replace the first generation of port-based tools [3]. Payload analysis [4], however, uses extensive computational resource [5]. This difficulty triggered the development of statistical analysis based approaches [6] often based on Machine-Learning Algorithms (MLAs) [7, 8] instead of direct payload inspection. These methods often rely on statistical protocol properties such as packet size and intra-packet arrival time distributions. Although some authors claim these algorithms provide high detection accuracy [9, 10], real-life tests [11–16] demonstrated that:

- Such algorithms are able to classify only a few traffic categories (an order of magnitude less than DPI libraries) and thus less suitable for fine protocol granularity detection applications.
- Some tests show a significant rate of inaccuracy suggesting that such methods may be useful in passive traffic analysis, but unlikely to be used for mission critical applications, such as traffic blocking.

These needs constitute the motivation for developing an efficient open-source DPI library where efficiency is defined by the requirement to monitor 10 Gbps traffic using solely commodity hardware (i.e., not specialized hardware needed). The use of open-source is essential because:

- Commercial DPI libraries are very expensive both in terms of one-time license fee and yearly maintenance costs. Sometimes their price is set based on yearly customers revenues, rather than on a fixed per-license fee, thus further complicating the price scheme.

- Closed-source DPI toolkits are often not extensible by end-users. This means that developers willing to add new/custom protocols support need to request these changes to the toolkits manufacturer. In essence, users are therefore at the mercy of DPI library vendors in terms of cost and schedule.
- Open-source tools cannot incorporate commercial DPI libraries as they are subject to a Non-Disclosure Agreement (NDA) that makes them unsuitable to be mixed with open-source software and included into the operating system kernel.

Although DPI was a hot topic for a long time, beside some rare exceptions, most research works did not lead to the creation of a publicly available DPI toolkit but limited their scope to prototypes or prof-of-concept tools. The need to create an efficient open-source DPI library for network monitoring was the motivation for this work.

The rest of the paper is structured as follows. Section 2 describes the motivation of this work and explains the differences from its predecessor OpenDPI [17]. Section 3 covers the nDPI design and implementation, while section 4 describes the validation process. Section 5 concludes the paper. The current work on nDPI and the future plans are described in Section 6.

2 Background and Motivation

DPI is defined as the analysis of a packet's data payload in real time (i.e., DPI processing must be faster than the traffic rate to be monitored as otherwise it would result in packet drops) at a given physical location. Inspection is motivated by various reasons including application protocol identification, traffic pattern analysis, and metadata (e.g., user name) extraction. Some proprietary DPI library vendors such as iPoque, QOSMOS, and Vineyard cover all aspects, whereas others, such as libprotoident [18], UPC [11], L7-filter [19], and TIE [20] limit their scope to protocol identification [21–23].

Protocol detection may also be implemented using pattern matching or by using specialized protocol decoders. The former approach is inefficient due to the use of regular-expressions [24] and error-prone because it does not reconstruct packets in 6-tuple flows (VLAN, Protocol, IP/port source/destination), thus missing cross-packet matches. Searching for patterns within an un-decoded payload can also lead to out of context search data (e.g., an email including an excerpt from a HTTP connection might be confused with web-traffic) or mismatches when specific packet fields (e.g., NetBIOS host name) are encoded.

Application drive the selection of the appropriate DPI library. We chose to focus on network traffic monitoring that can range from passive packet analysis to active inline packet policy enforcement. A DPI library must include the following features:

- High-reliability protocol detection for inline, per application, protocol policy enforcement.

- Library extensibility is needed for new protocols and runtime in sub-protocols definition. This feature is required because new protocols appear from time to time or evolve (e.g., the Skype protocol changed significantly since after the Microsoft acquisition). Permanent library maintenance is, therefore, required.
- Ability to integrate under an open-source license for use by existing open-source applications and embedding into an operating system's kernel. As already discussed, full source code availability is essential to safeguard privacy.
- Extraction of basic network metrics (e.g., network and application latency) and metadata (e.g., DNS query/response) that can be used within monitoring applications thus avoiding duplicate packet decoding, once in the DPI library and also in the monitoring application.

Our focus is, therefore, reliable protocol detection using protocol decoders combined with the ability to extract selected metadata parameter for the use of applications that is this library. This enables the extraction of selected metadata parameters that can then be used by applications using the DPI library. Other open-source protocol detection libraries, such as libprotoident, are limited in scope because they do not extract metadata and only analyze the first 4 B of payload in each direction for protocol detection. Because commercial DPI libraries could not be used a starting basis, we chose OpenDPI, an open-source predecessor of the commercial iPoque Protocol and Application Classification Engine (PACE), which is no longer maintained by its developers. OpenDPI was designed to be both an application protocol detection and metadata extraction library. Because it was unmaintained for some time, the library did not include any modern protocol (e.g., Skype); the code was largely prototype quality and likely used as a proof of concept for the commercial product. A point in favor of OpenDPI was the fact that it was distributed under the GPLv3 license that allows developers to include it in software applications without being bound to an NDA or other restrictions typical of commercial DPI products. Furthermore, an open-source license allows the code to be inspected, key requirement when the packet payload is inspected and potentially private information might leak. Our choice of OpenDPI as the starting point was driven by these reasons.

2.1 From OpenDPI to nDPI

The OpenDPI library is written in C and it is divided in two main components: the core library (responsible for handling raw packets, decoding IP layers 3 and 4, and extracting basic information such as IP address and port) and plugin dissectors (responsible for detecting the ~100 protocols supported by OpenDPI). nDPI inherited this two-layer architecture but it addressed several issues present in the OpenDPI design:

- The OpenDPI library was designed to be extensible, but in practice the data structures used internally were static. For instance, many data-types and bitmaps,

used to keep the state for all supported protocols, were bound to specific sizes (e.g., 128 bits) and thus limiting the number of identifiable protocols.

- Whenever a protocol was detected, the library tried to find further protocol matches instead of just returning the first match. The result was a performance penalty without a real need of requiring extra detection work.

- No encrypted protocol support (e.g., HTTPS). While encryption is designed to preserve privacy and regular DPI libraries are not expected to decode the some information can be gleaned to suggest the nature of the information carried on a specific connection.

- OpenDPI was not designed to be a reentrant (i.e., thread-safe) library as it used shared global variables. This required multi-threaded applications to create several instances of the library or add semaphores in order to avoid multiple threads to modify the same data at the same time. Per thread library state was required to support reentrancy.

- Many parts of OpenDPI suggest problematic design choices. For instance, the library was performing per-flow initializations, instead of doing them at once. As the result, the applications using the library paid an unnecessary performance penalty whenever a new connection was passed to OpenDPI.

- The protocol dissection was non-hierarchical. In other words, whenever a new connection needed to be analyzed, the library was not applying the dissectors based on their matching probability. For instance, if there is a connection on TCP port 80, OpenDPI was not trying the HTTP dissector first, but it was applying dissectors in the same order as they were registered in the library.

- The library had no runtime configuration capability; the only way to define new dissectors was to code them in C. While this is usually a good policy for efficiency reasons, at times more flexibility is needed. For instance, if a given user needs to define a custom protocol Y as TCP/port-X it would be easier to have a runtime configuration directive instead of changing the library code. OpenDPI assumes that the library must have a dissector for all supported protocols, a difficult goal to achieve in reality. In particular, in closed-environments such as a LAN, specific hosts use proprietary/custom protocols that flow on specific ports/protocols. In this case, it is more convenient for the user to detect them from the packet header rather than from its payload.

- OpenDPI was not designed to extract any metadata from analyzed traffic. On one hand this preserves privacy, but on the other it requires monitoring applications to decode the application traffic again in order to extract basic information such

as the URL from HTTP traffic. Reporting this information does not add any overhead to the library as it is decoded anyway when parsing the packet payload.

Summarizing, OpenDPI was a good starting point for nDPI, because we did not have to start from scratch. Many components of the original library were changed in order to address the issues we identified. This was the ground work necessary to start the creation of an efficient DPI library and extending the set of supported protocols. Not surprisingly, the number of protocols recognized has an impact on both DPI detection performance and protocol recognition. The more protocols recognized, the more time spent on detection whenever a specific traffic pattern is not identified and thus all the possible protocol decoders have to be tested for match. This means that DPI libraries supporting many protocols may be slower in specific situation than those supporting fewer. Another impact on performance is due to metadata extraction: the richer the set of extracted attributes, the slower the processing. Although specific activities such as string and pattern matching can be accelerated on specialized hardware platforms such as Cavium and RMI, or using GPUs [25], we decided not to use any of these cards, in order to let the library operate on all hardware platforms.

nDPI was designed to be used by applications that need to detect the application protocol of communication flow. Its focus is on Internet traffic, thus all the available dissectors support standard protocols (e.g., HTTP and SMTP) or selected proprietary ones (e.g., Skype or Citrix) that are popular across the Internet community. In the latter case, as protocol specifications are not publicly available, we had to create the dissectors by reverse-engineering network traffic. Although nDPI can extract specific metadata (e.g., HTTP URL) from analyzed traffic, it was not designed as a library to be used in fields such as lawful interception or data leak prevention; its primary goal is to characterize network traffic. Similarly to OpenDPI, nDPI can be used both inside the Linux kernel and in user-space applications, and it work on most operating systems including Linux, Windows, MacOS X, as well as non-Intel CPU architectures such as ARM and MIPS.

3 Design and Implementation of nDPI

nDPI defines an application protocol by a unique numeric protocol Id and a symbolic protocol name (e.g., Skype). Applications using nDPI will probably use the protocol Id whereas humans the corresponding name. The protocol set includes both network protocols such as SMTP or DNS, and communications over network protocols. For instance, Facebook and Twitter are treated as two protocols, although that in fact they are communications from/to Facebook/Twitter servers. A protocol is usually detected by a traffic dissector written in C, but it can be defined also in terms of protocol/port, IP address (e.g., traffic from/to specific networks), and protocol attributes. For instance, the Dropbox traffic is identified by both the dissector for LAN-based communications,

and by tagging as Dropbox the HTTP traffic on which the 'Host' header field is set to '*.dropbox.com'. As explained later in this section, nDPI supports over 170 protocols, but it can also be further extended at runtime using a configuration file.

The nDPI library inherits some of OpenDPI design, where the library code is used for implementing general functions, and protocol dissection is implemented in plugins. All the library code is now fully reentrant, meaning that applications based on nDPI do not need to use locks or other techniques to serialize operations. All the library initialization is performed only once at startup, without a runtime penalty when a new packet needs to be dissected. nDPI expects the caller to provide the packet divided in flows (i.e., set of packets with the same VLAN, protocol, IP/port source/destination), and that the packet was decoded up to layer 3. This means that the caller has to handle all the layer-2 encapsulations, such as VLAN and MPLS, by leaving to nDPI the task of decoding the packet from the IP layer up. nDPI comes with a simple test application named pcapReader.c [26], that shows how to implement packet classification and provides utility functions for efficient flow processing. The protocol dissectors are registered with attributes, such as the default protocol and port. For instance, the HTTP dissector specifies the default TCP/80, and the DNS dissector TCP/UDP on port 53. This practice has two advantages:

- Packets belonging to a yet unclassified flow are passed to all registered dissectors, starting from the most likely one. For instance, a TCP packet on port 80 is first passed to the HTTP protocol and then (if not identified) to the remaining registered dissectors. Of course only dissectors for TCP protocols are considered, which reduces the number of dissectors that are tested, and decreases the matching time. This optimization does not prevent detecting HTTP on non-standard ports.
- When a flow is still unclassified (none of the dissectors matched), nDPI can guess the application protocol based on the transport protocol and port. Note that a flow can be unclassified not just because of protocol dissectors limitations, but also because not all flow packets where passed to nDPI. A typical example is the case when nDPI has to dissect packets belonging to a flow whose beginning was not analyzed (e.g., nDPI was activated after the flow start).

The protocol recognition lifecycle for a new flow begins from decoding layers 3 and 4 of the packet. In case there is a dissector registered for the packet protocol/port, such dissector is tried first. In case of no match, all the compatible dissectors (i.e., in case of a UDP packet, all UDP dissectors) are tried. If a dissector cannot match a packet, it has two options: either the match failed because the analyzed packet will never match (e.g., a DNS packet passed to the SNMP dissector), or it failed but it may be that future packets will match. In the former case, the dissector will not be considered for future packets belonging to the same flow, whereas in the latter the dissector will still be considered for future packets belonging to the same flow. Protocol detection ends as soon as a dissector matches.

A typical question asked by nDPI users is how many packets are needed to detect the application protocol or to decide that the given flow is unknown. From our experience, the number is protocol dependent. For most UDP-based protocols, such as DNS, NetFlow, or SNMP one packet is enough to make this decision. Unfortunately, there are other UDP-based protocols such as BitTorrent, whose signature might require up to 8 packets in order to be detected. This leads us to the rule of thumb that for nDPI at most 8 packets per direction are enough to make a decision.

3.1 Handling Encrypted Traffic

The trend of Internet traffic is towards encrypted communications. Due to security and privacy concerns, HTTPS is slowly replacing HTTP not just for secure transactions but also for sending tweets and messages to mobile terminals, posting notes, and performing searches. Identifying this traffic as SSL is not enough, but it is necessary to better characterize it. When using encrypted communications, the only part of the data exchange that can be decoded is the initial key exchange. nDPI contains a decoder for SSL that extracts the host name from the server certificate. This information is placed in the nDPI flow metadata in a similar way as the server name from the 'Host:' HTTP header. With this approach we can identify known services and tag them according to the server name. For instance, an encrypted communication towards a server named 'api.twitter.com' is marked as Twitter, 'maps.google.com' as Google maps, and '*.whatsapp.net' as the WhatsApp messaging protocol. We can also discover self-signed SSL certificates, which is important as it might indicate that the connection is not safe, not just in terms of data leak, but also in terms of the activity behind the communication. For instance symmetric (i.e., the traffic is not predominant in one direction such as in HTTPS, where the client sends little traffic with respect to the traffic sent by the server) long standing SSL connections with self-signed certificates often hide SSL VPNs.

As described later in this section, nDPI contains internally a configuration for many known protocols that are discovered using the above technique. In addition, it is possible to add at runtime a configuration file that further extends the set of detected protocols so that new ones can be defined without changing the protocol dissector. With the advent of Content Delivery Networks (CDN) this is probably the only way of identifying the application protocol, as at any given time the same server (identified with a single IP address) can deliver two different services provided by two customers using the same CDN. As a fallback, nDPI can identify specific application protocols using the IP address. For instance, nDPI detects many Apple-provided services, such as iTunes and iMessage. In addition to that, it marks as Apple (a generic protocol) all communications that were not specifically identified, but that were exchanged with the Apple-registered IP addresses (i.e., 17.0.0.0/8).

3.2 Extending nDPI

As previously explained, nDPI users can define protocols not just by adding a new protocol dissector, but also providing a configuration file at runtime. New protocols are defined by name. When nDPI detects that a protocol name is already defined (e.g., in the above example SIP and HTTP are handled by the native dissector), the configuration file extends the default configuration already present in nDPI. For instance in the previous example, whenever nDPI sees TCP traffic on port 81 or 8181 it tags it as HTTP. Additionally, nDPI can also identify a protocol using strings that are matched against metadata extracted from the nDPI flow such as HTTP Host and SSL certificate server name. The defined strings are stored by an automata based on the Multifast [27] library that implements string matching according to the Aho-Corasick algorithm. This library is quite efficient: at startup the automata creation takes little time (i.e., almost instantaneous with tenth of strings, or some seconds with hundred thousand strings), then this library configured performs over 10 Gbps during search when configured with hundred thousand strings.

4 Validation of nDPI

There are recent papers that compare the nDPI accuracy in terms of protocol detection against other DPI toolkits. Their conclusion is that "nDPI and libprotoident were successful at correctly classifying most (although admittedly not all) of the applications that we examined and only one of the evaluated applications could not be classified by both tools" [14], and "the best accuracy we obtained from nDPI (91 points), PACE (82 points), UPC MLA (79 points), and Libprotoident (78 points)" [11]. These tests showed that nDPI is pretty accurate, even more accurate than PACE, the commercial version of the old OpenDPI library on which nDPI is based. We are aware that nDPI had some false positives with Skype and BitTorrent due to the use of heuristics. In the further nDPI versions (svn revision 7249 or newer), we decided to remove the use of these heuristics, so that we basically eliminated false positives at the cost of slightly increasing the number of undetected flows when using these two protocols. We also re-implemented support for several other protocols, as FTP, SOCKSv4, SOCKSv5, eDonkey, PPLive, Steam, RTMP, and Pando. Furthermore, we added the ability to discover new web services, e.g., Wikipedia and Amazon.

The latest version of nDPI was tested using the same methodology and the same dataset as in [28], so we could compare the results and show the impact of the latest changes. In essence, the ground-truth was established by a host-based traffic monitoring system, which marks flows by the process names obtained from the system sockets. The results in the term of percent of flows classified correctly, incorrectly, and left unclassified for the most popular protocols, applications, and web services are shown in Table 1. Sometimes, the results were obtained on another level than the test presumed. For

Table 1: Evaluation of the Latest Version of nDPI

Protocol	Flows	Correct [%]	Wrong [%]	Unknown [%]
DNS	18 251	100.00	0.00	0.00
HTTP	42 983	97.80	0.66	1.54
ICMP	205	100.00	0.00	0.00
IMAP (Start-TLS)	35	100.00	0.00	0.00
IMAP (TLS)	103	100.00	0.00	0.00
NETBIOS (Name Service)	10 199	99.97	0.00	0.03
NETBIOS (Session Service)	11	100.00	0.00	0.00
Samba Service	42 808	100.00	0.00	0.00
NTP	42 227	100.00	0.00	0.00
POP3 (plain mode)	26	100.00	0.00	0.00
POP3 (TLS)	101	100.00	0.00	0.00
RTMP	378	70.90	15.87	13.23
SMTP (plain mode)	67	100.00	0.00	0.00
SMTP (TLS)	52	100.00	0.00	0.00
SOCKSv5	1 927	92.99	0.00	7.01
SSH	38 961	93.98	0.80	5.22
BitTorrent (encrypted mode)	96 399	54.41	0.18	45.41
BitTorrent (mixed mode)	261 527	99.41	0.02	0.57
DropBox	93	98.92	0.00	1.08
eMule (obfuscated)	12 835	11.04	2.67	86.29
eMule (mixed mode)	13 852	17.57	2.28	80.15
FTP (active mode)	126	98.41	0.00	1.59
FTP (passive mode)	122	72.95	0.00	27.05
iTunes	235	93.62	0.00	6.38
Pando Media Booster	13 453	99.26	0.63	0.11
PPLive	1 510	43.91	1.06	55.03
RDP	153 837	99.69	0.02	0.29
Skype	2 177	92.38	7.44	0.18
Sopcast	424	63.68	8.49	27.83
Steam	1 205	76.02	0.42	23.57
TOR	185	33.51	0.00	66.49
web: Amazon	602	83.89	0.00	16.11
web: Apple	477	74.63	0.00	25.37
web: Facebook	6 953	80.14	0.00	19.86
web: Google	6 541	83.03	0.02	16.95
web Wikipedia	6 092	68.96	0.23	30.81
web: Yahoo!	17 373	83.16	0.02	16.82
web: YouTube	2 534	82.16	0.00	17.84

```
# taskset -c 1 ./pcapReader -i ~/test.pcap
Using nDPI (r7253)
pcap file contains
IP packets: 3000543 of 3295278 packets
IP bytes:1043493248(avg pkt size 316 bytes)
Unique flows: 500
nDPI throughput: 3.42 M pps / 8.85 Gb/sec
```

Figure 1: Test Outcome of nDPI Validation

example, when we tested for the ability to detect the HTTP traffic, we also obtained results as *Google*, *Facebook*, or *DropBox*. In this evaluation, we acknowledged such results as correct, even though the protocol was not explicitly given. That could lead to a minor inaccuracy in case if the detected *Facebook* traffic was in fact not HTTP (but, for example, SSL). This is the only difference from the tests performed in [28], where only the results on the asked level were considered as correct (e.g., *HTTP*), while the others (e.g., *Facebook*) were considered as being unclassified).

As there are many extensive tests on nDPI protocol detection accuracy, this paper focuses on nDPI performance. For that purpose, we developed an application named pcapReader [26] that can both capture from a physical network device and read packets from a pcap file. To test nDPI on a physical network at 10 Gbps, we used the test application on top of PF_RING [29], which allows applications on commodity hardware to process packets in RX/TX at 10 Gbps line rate for any packet size. For our tests, we used a pcap file of over 3 million packets, captured on a heterogeneous environment, thus including both LAN protocols (e.g., NFS and NetBios) and Internet protocols (e.g., Skype and DropBox). We used a PC running Ubuntu Linux 13.10 (kernel 3.11.0-15) on a 8 core Intel i7 860. We bound the application to a single core, in order to test it in the worst case, and see how the application can scale when using multiple cores. The test outcome is depicted in Figure 1.

The outcome demonstrated that the test application processes packets at an average speed of 3.5 Mpps / 8.85 Gbps using a single core. As the test pcap file used during the test was captured on a real network, it contained some flows that began before the packet capture started. nDPI detects a flow protocol by looking at the initial flow packets, so some flows are not detected due to this reason. For undetected flows, nDPI can guess the protocol by using the flow protocol/port registered during startup or it can leave the flows undetected. When using this test application over PF_RING DNA on a 10 Gbps Intel adapter, it is possible to use the network driver with hardware flow balancing. In this way, we can start one instance of the test application per virtual queue, binding each instance to a different core. In sum, 10 Gbps traffic can be inspected when balanced across two cores (the above tests show DPI at 8 Gbps using a single core), using the modestly priced commodity hardware we used in our tests.

In terms of memory usage, nDPI needs some memory to load the configuration and automatas used for string-based matching. This memory used by nDPI is ~210 KB with no custom configuration loaded that increases of ~25 KB when a custom configuration is loaded. In addition to that, nDPI keeps per-flow information that is independent from the application protocols and which takes ~1 KB per flow.

5 Conclusion

This paper presents nDPI, an open-source toolkit released under GPLv3 license available at https://svn.ntop.org/svn/ntop/trunk/nDPI/. It is currently able to detect more than 170 protocols including Skype, BitTorrent, and other messaging protocols. The validation test performed by third parties demonstrated that nDPI outperforms some commercial and open-source toolkits in terms of protocol recognition accuracy. In terms of performance, using two CPU cores and commodity hardware, nDPI can handle a 10 Gbit link fully loaded with Internet traffic. This makes it suitable for scenarios where both detection accuracy and high performance are a requirement.

6 Final Remarks

The development of nDPI is still ongoing. There are several important features, which are missing in the current version of nDPI, however, they are planned to be included in the official SVN trunk in the near future.

The most important drawback of the current implementation (as well as of the majority of the currently existing traffic classification tools) is the quality of the provided results. Habitually, the classifiers provide only result per flow, which is supposed to characterize the flow in the most detailed manner. Therefore, the output is a mix of results on various levels: IP protocols (i.e., TCP or UDP), application protocols (e.g., DNS, HTTP, SSL, BitTorrent, or SMTPS), types of the content (e.g., MPEG, or Flash), and finally, the service providers (e.g., Facebook, or YouTube). The usefulness of such result is very limited. At first, one flow can use several application protocols (as HTTP and Dropbox). At second, one application protocol (as DNS) can use several IP protocols (TCP and UDP). At third, it is impossible to judge if the most detailed level is the content (as Flash) or the service (as YouTube). Finally, this scheme does not allow to provide precise accounting of the traffic (for example, it is not possible to account the HTTP traffic, if the results are given on multiple levels, so in majority of cases HTTP is even not mentioned). Therefore, we are changing the format of the results, so all the possible classifications will be consistently provided.

References

[1] Network Based Application Recognition (NBAR) – Cisco, 2009. [Online]. Available: http://www.cisco.com/c/en/us/products/ios-nx-os-software/network-based-application-recognition-nbar/index.html.

[2] Palo Alto Networks, Next-Generation Firewall Overview, 2011. [Online]. Available: https://www.paloaltonetworks.com/content/dam/paloaltonetworks-com/en_US/assets/pdf/datasheets/firewall-features-overview/firewall-features-overview.pdf.

[3] Marco Mellia, Antonio Pescapè, and Luca Salgarelli. Traffic classification and its applications to modern networks. *Computer Networks*, 53(6):759–760, April 2009. DOI: 10.1016/j.comnet.2008.12.007.

[4] Fulvio Giovanni Risso, Mario Baldi, Olivier Morandi, A Baldini, and P. Monclus. Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation. In *ICC'08. IEEE International Conference on Communications, 2008*, pages 5869–5875. IEEE, Beijing, China, May 2008. DOI: 10.1109/ICC.2008.1097.

[5] Matteo Avalle, Fulvio Risso, and Riccardo Sisto. Efficient multistriding of large non-deterministic finite state automata for deep packet inspection. In *2012 IEEE International Conference on Communications (ICC)*, pages 1079–1084. IEEE, Ottawa, Ontario, Canada, June 2012. DOI: 10.1109/ICC.2012.6364235.

[6] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16, January 2007. DOI: 10.1145/1198255.1198257.

[7] Tomasz Bujlow, Tahir Riaz, and Jens Myrup Pedersen. A method for classification of network traffic based on C5.0 Machine Learning Algorithm. In *Proceedings of ICNC'12: 2012 International Conference on Computing, Networking and Communications (ICNC): Workshop on Computing, Networking and Communications*, pages 244–248. IEEE, Maui, Hawaii, USA, February 2012. DOI: 10.1109/ICCNC.2012.6167418.

[8] Thuy T. T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4):56–76, October 2008. DOI: 10.1109/SURV.2008.080406.

[9] Sven Ubik and Petr Žejdl. Evaluating application-layer classification using a Machine Learning technique over different high speed networks. In *Proceedings of the Fifth International Conference on Systems and Networks Communications (ICSNC)*, pages 387–391. IEEE, Nice, France, August 2010. DOI: 10.1109/ICSNC.2010.66.

[10] Li Jun, Zhang Shunyi, Lu Yanqing, and Zhang Zailong. Internet Traffic Classification Using Machine Learning. In *Proceedings of the Second International Conference on Communications and Networking in China (CHINACOM '07)*, pages 239–243. IEEE, Shanghai, China, August 2007. DOI: 10.1109/CHINACOM.2007.4469372.

[11] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification. Technical report, Department of Computer Architecture (DAC), Universitat Politècnica de Catalunya (UPC), June 2013. Accessible: https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2013,en.html.

[12] Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Quantifying the accuracy of the ground truth associated with Internet traffic traces. *Computer Networks*, 55(5):1158–1167, April 2011. DOI: 10.1016/j.comnet.2010.11.006.

[13] Niccolò Cascarano, Luigi Ciminiera, and Fulvio Risso. Optimizing deep packet inspection for high-speed traffic analysis. *Journal of Network and Systems Management*, 19(1):7–31, March 2011. DOI: 10.1007/s10922-010-9181-x.

[14] Alcock, Shane and Nelson, Richard. Measuring the Accuracy of Open-Source Payload-Based Traffic Classifiers Using Popular Internet Applications. In *IEEE Workshop on Network Measurements (WNM), the 38th IEEE Conference on Local Computer Networks (LCN)*. IEEE, Sydney, Australia, October 2013.

[15] Ryan Goss and Reinhardt Botha. Deep Packet Inspection – Fear of the Unknown. In *Information Security for South Africa (ISSA), 2010*, pages 1–5. IEEE, Sandton, Johannesburg, South Africa, August 2010. DOI: 10.1109/ISSA.2010.5588278.

[16] Silvio Valenti, Dario Rossi, Alberto Dainotti, Antonio Pescapè, Alessandro Finamore, and Marco Mellia. Reviewing Traffic Classification. In *Data Traffic Monitoring and Analysis*, pages 123–147. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-36784-7_6.

[17] Yong Wei, Yun-Feng Zhou, and Li-chao Guo. Analysis of Message Identification for OpenDPI. *Computer Engineering*, (2011-S1), 2011. Accessible: http://en.cnki.com.cn/Article_en/CJFDTOTAL-JSJC2011S1033.htm.

[18] Shane Alcock and Richard Nelson. Libprotoident: Traffic Classification Using Lightweight Packet Inspection. Technical report, University of Waikato, August 2012. Accessible: http://www.wand.net.nz/publications/lpireport.

[19] Application Layer Packet Classifier for Linux, 2009. [Online]. Available: http://l7-filter.sourceforge.net/.

[20] Alberto Dainotti, Walter de Donato, and Antonio Pescapé. Tie: A community-oriented traffic classification platform. In *First International Workshop on Traffic Monitoring and Analysis, TMA 2009, Traffic Monitoring and Analysis*, pages 64–74. Springer Berlin Heidelberg, Aachen, Germany, May 2009. DOI: 10.1007/978-3-642-01645-5_8.

[21] Amir R. Khakpour and Alex X. Liu. High-speed flow nature identification. In *29th IEEE International Conference on Distributed Computing Systems, 2009 (ICDCS'09)*, pages 510–517. IEEE, Montreal, Quebec, Canada, June 2009. DOI: 10.1109/ICDCS.2009.34.

[22] Hyunchul Kim, Kimberly C. Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In *Proceedings of the 2008 ACM CoNEXT conference*, page 8. ACM New York, Madrid, Spain, December 2008. DOI: 10.1145/1544012.1544023.

[23] Giuseppe Aceto, Alberto Dainotti, Walter De Donato, and Antonio Pescapé. PortLoad: taking the best of two worlds in traffic classification. In *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*, pages 1–5. IEEE, San Diego, California, USA, March 2010. DOI: 10.1109/INFCOMW.2010.5466645.

[24] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. Algorithms to accelerate multiple regular expressions matching for Deep Packet Inspection. *ACM SIGCOMM Computer Communication Review – Proceedings of the 2006 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM '06)*, 36(4):339–350, October 2006. DOI: 10.1145/1159913.1159952.

[25] Niccolo' Cascarano, Pierluigi Rolando, Fulvio Risso, and Riccardo Sisto. iNFAnt: NFA pattern matching on GPGPU devices. *ACM SIGCOMM Computer Communication Review*, 40(5):20–26, October 2010. DOI: 10.1145/1880153.1880157.

[26] ntop – PCAP Reader, 2014. [Online]. Available: `https://svn.ntop.org/svn/ntop/trunk/nDPI/example/pcapReader.c`.

[27] Multifast 1.4.2, 2014. [Online]. Available: `http://multifast.sourceforge.net`.

[28] Tomasz Bujlow, Valentín Carela-Español, and Pere Barlet-Ros. Extended Independent Comparison of Popular Deep Packet Inspection (DPI) Tools for Traffic Classification. Technical report, Department of Computer Architecture (DAC), Universitat Politècnica de Catalunya (UPC), January 2014. Accessible: `https://www.ac.upc.edu/app/research-reports/html/research_center_index-CBA-2014,en.html`.

[29] Francesco Fusco and Luca Deri. High speed network traffic analysis with commodity multi-core systems. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC '10)*, pages 218–224. ACM, Melbourne, Australia, November 2010. DOI: 10.1145/1879141.1879169.

Classification and Analysis of Computer Network Traffic

TOMASZ BUJLOW

Traffic monitoring and analysis can be done for multiple different reasons: to investigate the usage of network resources, adjust Quality of Service (QoS) policies in the network, log the traffic to comply with the law, or create realistic models of traffic for academic purposes. The core activity in this area is traffic classification, which is the main topic of this thesis.

We introduced the already known methods for traffic classification (as by using transport layer port numbers, Deep Packet Inspection (DPI), statistical classification) and assessed their usefulness in particular areas. Statistical classifiers based on Machine Learning Algorithms (MLAs) were shown to be accurate and at the same time they do not consume a lot of resources and do not cause privacy concerns. However, they require good quality training data. We performed substantial testing of widely used DPI classifiers and assessed their usefulness in generating ground-truth, which can be used as training data for MLAs. Because the existing methods were shown to not be capable of generating the proper training data, we built our own host-based system for collecting and labeling of network data, which depends on volunteers. Afterwards, we designed and implemented our own system for traffic classification based on various statistical methods, which provides consistent results on all of the 6 levels: Ethernet, IP protocol, application, behavior, content, and service provider. Finally, we contributed to the open source community by improving the accuracy of nDPI traffic classifier. The thesis also evaluates the possibilities of using various traffic classifiers in order to assess the per-application QoS level.

2014
Aalborg University
Department of Electronic Systems
Networking & Security
www.es.aau.dk/netsec
ISBN: 978-87-71520-30-9

AALBORG UNIVERSITY
DENMARK